68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $9.45 | Hong Kong H $23.50
Malaysia M $9.45 | Sweden 30:-SEK

$2.95 USA

OS-9 Atari Amiga Mac S-50

6800 6809 68008 68000 68010 68020 68030

The Magazine for Motorola CPU Devices *For Over a Decade!*

This Issue:

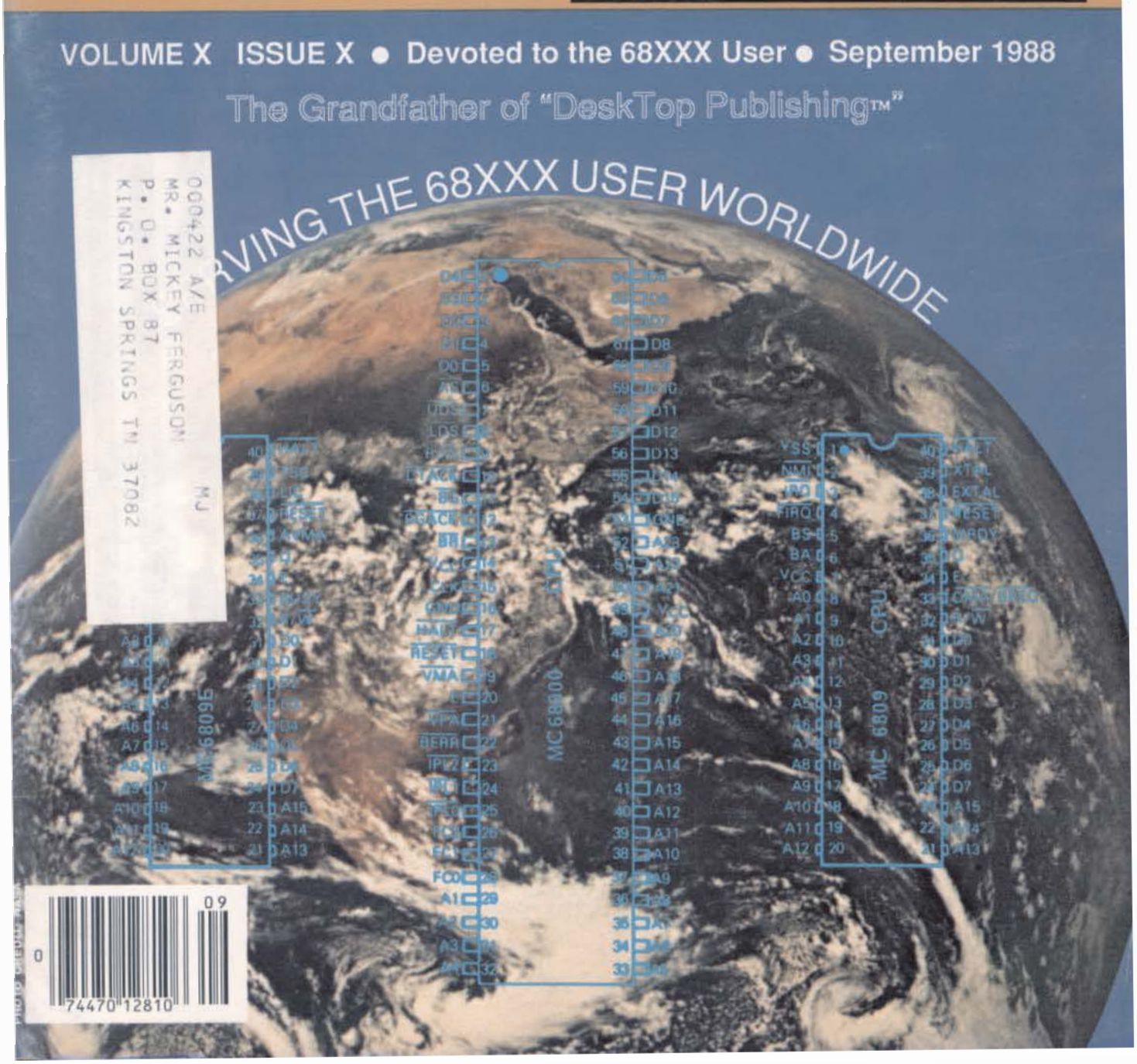
OS-9 SK*DOS Atari Amiga FLEX Macintosh

A User Contributed Journal

And Lots More!

VOLUME X ISSUE X • Devoted to the 68XXX User • September 1988

The Grandfather of "DeskTop Publishing™"



0 74470 12810 09

WHO DO YOU CALL WHEN YOUR DEBUGGER WON'T DEBUG?

The problem with most real-time operating systems is simple, they're not an integrated solution. You end up dealing with a multitude of suppliers for languages, compilers, debuggers and other important development tools. And when something does go wrong, it can be a frustrating experience trying to straighten out the mess.

Why Not Try the Microware One-Stop Total Solution?

Microware's OS-9 Real-Time Operating System is a total integrated software system, not just a kernel. We offer an extensive set of development tools, languages, I/O and Kernel options. ***And this total integrated solution is entirely designed, built and supported by the same expert Microware team.***

Modularity Lets YOU Choose Just What You Need.

The modular design of OS-9 allows our Operating System to adapt as your requirements change. OS-9 can support a complete spectrum of applications — from embedded ROM-based code in board-level products all the way up to large-scale systems.

The OS-9 Success Kit

A Total Integrated Solution for Your Next Project

Development Tools:
- C Source Level Debugger
- Symbolic Debugger
- System State Debugger
- uMACS Text Editor
- Electronic Mail
- Communications
- Super Shell

Kernel Options:
- MMU (Security Protection) Support
- Math Coprocessor Support

Languages:
- C*
- Basic
- Pascal
- Fortran
- Ada**
- Assembler*

I/O Options:
- SCSI, SASI & SMD Disks
- 3-, 5-, 8-inch Diskettes
- Magnetic Tape
- Ethernet - TCP/IP
- Arcnet - OS-9/Net

*Resident or UNIX versions available
**VAX hosted

Support is Part of the Package.

Microware is proudly setting the industry's standard for customer support. You'll find professional and comprehensive technical documentation and a Customer Hotline staffed by courteous and authoritative software engineers.

So stop messing with simple kernels and independent suppliers. Call Microware today and find out more about the "One-Stop Integrated Solution" with OS-9!

microware® OS-9

Microware Systems Corporation
1900 N.W. 114th Street
Des Moines, Iowa 50322
Phone: 515/224-1929

Western Regional Office
4401 Great America Parkway
Santa Clara, California 95054
Phone: 408/980-0201

Microware Japan Ltd.
41-19 Ioncho 4-Chome
Funabashi City
Chiba 273, Japan
Phone: 0474 (22) 1747

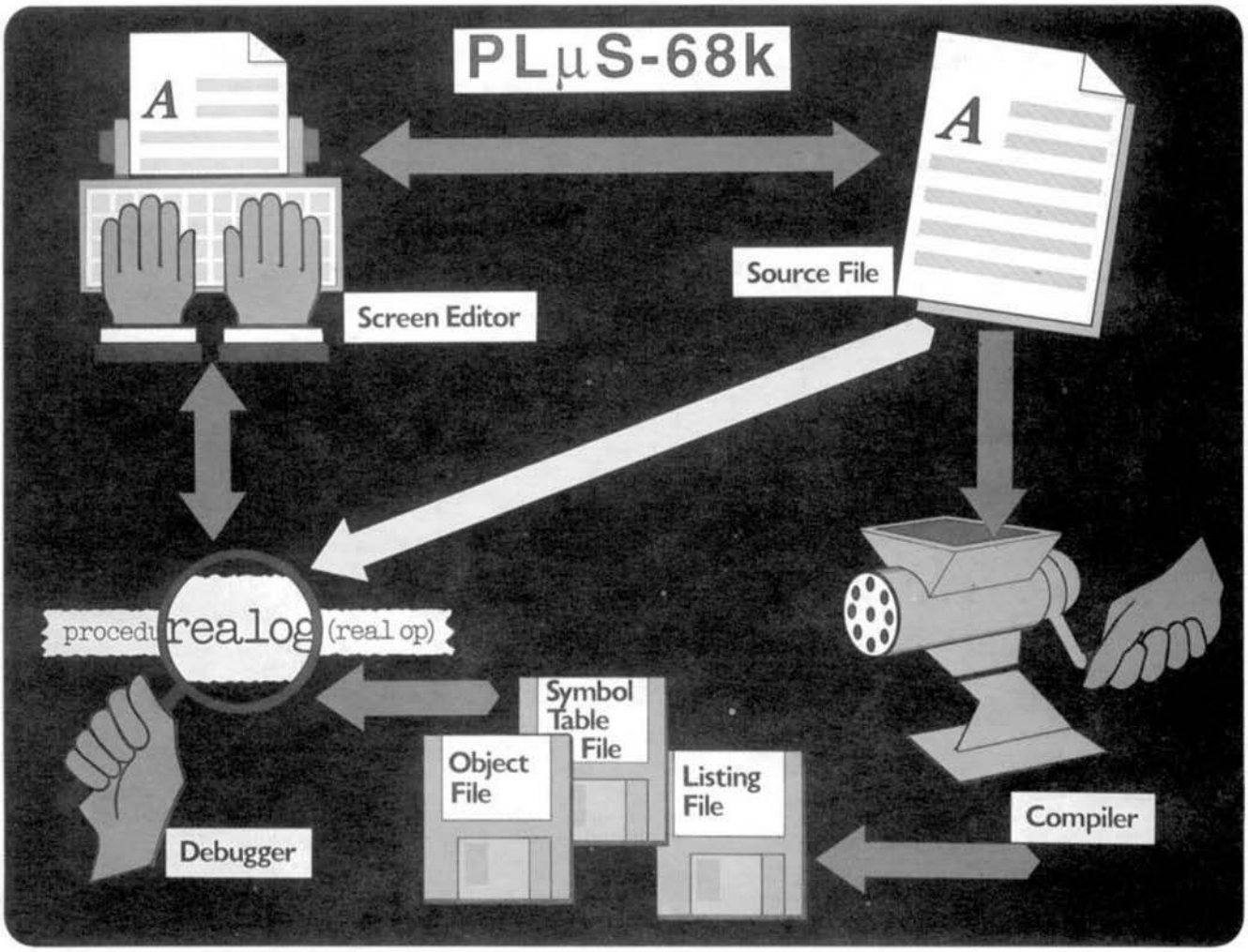
PLμS-68k

A Complete Program Development Environment

MC68000 - MC68010 - MC68020/881 Processors

OS-9 User-state or System-state Modules

Stand Alone ROM Based Targets



A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Cheryl Hodge

Contributing & Associate Editors

Ron Anderson
Ron Voigts
Doug Lurie
Ed Law
Dr. E.M. "Bud" Pass
Art Weller
Dr. Theo Elbert
& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343
Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

---

68 Micro Journal (ISSN 0194-5025) is published 12 times a year by Computer Publishing Inc. Second Class Postage paid at Hixson, TN. and additional entries. POSTMASTER: send address changes to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

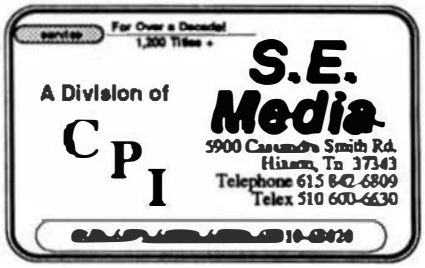
PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems



PAT FROM S. E. MEDIA -- A FULL FEATURED **SCREEN ORIENTED TEXT EDITOR** with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included**. Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

MUSTANG-020 Super SBC™

DATA-COMP Proudly Presen s the First Under $4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under $2400 complete!

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with r/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

Available 12.5- 25 Mhz systems, call for special prices

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

Data-Comp Division

CPI A Decade of Quality Service"

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.32*

```
Main()
        {
            register long i;
            for (i=0; i < 999999; ++i);
        }
```

Estimated MIPS - MUSTANG-020 — 6.5 MIPS,
Burst to 8 - 10 MIPS: Motorola Spec

OS-9

OS-9 Professional Ver	$850.00
*Includes C Compiler	
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Forven 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph.Spell.Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Forven 77	450.00
Sculptor+ (see below)	995.00
Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adaptor Cards below RS232 Adaptor	165.00
Each card supports 4 additional ser. ports (total of 36 serial ports supported)	
60 line Parallel I/O card Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	398.00
Prototype Board area for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	75.00
SBC-AN Interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call.	475.00
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00
Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

The PRO!

Mustang Specifications

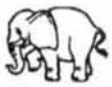
12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
Tered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

CPI™
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead!
ONLY Data-Comp delivers the Super MUSTANG-020

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Only the "PRO" Version of OS-9 Supported!



This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

Save $1000.00
Complete
25 Mbyte HD System
$4299.80
85Mbyte HD System
$5748.80

Note: Only Professional OS-9 Now Available (68020 Version)
Includes ($500) C Compiler - 68020 & 68881 Supported -
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division
A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL 60139

MORE DEVICES

If you read the last column, you are probably asking what about the SCFMan type devices? Last time covered briefly device managers and drivers. I talked about device descriptors touching on the RBF typedevices. But nothing furtherwas saidabout SCF device descriptors. Well that is this months topic.

An SCF device is one which receives its input and sends it output a byte at a time. Rather then move the blocks of data like the RBF devices, it works on a byte-by-byte bases. This type of data flow is perhaps the easiest and yet is efficient. SCF devices include items like terminals, printers, video screens and keyboards.

Last month I gave a sample assembly language program and showed how to write a device descriptor for an RBF device. This month I present the SCF version in the Listing. As will be seen, it is not really necessary to write one for SCF devices. But I created this one for illustrative purposes. It is for /TERM on the Level 2 Color Computer 3.

The header is similar to last month's RBF version. There are a few differences. The device capabilites is %00000011 or $03. This is for read and write only. Obviously we could not make a directory out of it nor would we want everyone tied into the same terminal. It is not shareable either.

The real changes take place in the initialization table. There are two parts to the table — the standard entries and the window entries.

Standard Entries

IT.DVC is the device type.
0...SCF
1...RBF
2...PIPE
4...SBF

IT.UPC is the character case.
0...upper and lower case
1...upper case only

IT.BSO is character backspace status.
0...backspace over characters
1...backspace while erasing

IT.DLO is the line delete status.
0...return to column 1 deleting line
1...return to column 1 leaving line

IT.EKO is echo characters to output.
0...echo on
1...echo off

IT.ALF is auto line feed.
0...auto line feed off
1...auto line feed on

IT.NUL is the number of null characters after end-of-line.

IT.PAU is the page pause status.
0...no page pause
1...page pause

IT.PAG is the number of lines per page.

IT.BSP is the backspace character. Usually it is $08 or ^H. This can be changed.

IT.DEL is the line delete character. Usually it is $18 or ^X. This can be changed.

IT.EOR is the the end-of-record character. Usually it is $0D or ^M. This can be changed.

IT.EOF is the end-of-file character. Usually it is $1B or the escape character. This can be changed.

IT.RPR is the reprint-line character. Usually it is $04 or ^D. This can be changed.

IT.DUP is the duplicate-last-line character. Usually it is $01 or ^A. This can be changed.

IT.PSC is the pause character. Usually it is $17 or ^W. This can be changed.

IT.INT is the interrupt character. Usually it is $03 or ^C. This can be changed.

IT.QUT is the quit character. Usually it is $05 or ^E. This can be changed.

IT.BSE is the backspace echo character. Usually it is $08 or ^H. This differs from the other IT.BSP, since this is what is sent back to the terminal or printer. This can be changed.

IT.OVF is the overflow character. Usually it is $07 or ^G. This is character that rings the bell.

IT.PAR is device parity byte. It is used to initialize a device control register when a path is opened to it.

IT.BAU is the baud rate. The following values are a possible set. It could be different depending on the device driver.
0...110
1...300
2...600
3...1200
4...2400
5...4800
6...9600
7...19200

IT.D2P is the attached device's name offset. The header already indicates the module name's offset. This one may be used for other purposes. It is a part of the PD.OPT section.

IT.XON is the X-ON character. For the Color Computer terminal it is not necessary and is set to $00, but for standard terminal it can be something else. Usually it is $11 or ^Q.

IT.XOFF is the X-OFF character. For the Color Computer terminal it also is not necessary and is set to $00, but for standard terminals it can be something else. Usually it is $13 or ^S.

At this point the standard initialization table is complete. For the Color Computer and other systems that use windows, more information may be necessary. Following this is the window information. I am including Color Computer values. At a future time I will talk in more details about windows.

Window Entries

IT.COL is the number of columns. Usually this is 32, 40 or 80.

IT.ROW is the number of rows. Usually this 16 or 24.

IT.WND is the window number. Usually this is a value of 0 through 1.

IT.VAL is the whether the rest of the data is valid.
0...data not valid
1...data valid

IT.STY is the window type. This can be graphics and non-graphics. The window size and so forth.
1...40-Column hardware screen
2...80-column hardware screen
3...640 X 192 two-color screen
4...320 X 192 four-color screen
5...640 X 192 four-color screen
6...320 X 192 sixteen-color screen

IT.CPX is the position in the X-axis.

IT.CPY is the position in the Y-axis.

IT.FGC is the foreground color.

IT.BGC is the background color.

IT.BDC is the border color.

The listing at the end of this month's column is presented as an illustration of what is a device descriptor is like. There is good news! It is not necessary to create one. A means has been provided to change the current values of a device like the terminal or the printer.

XMODE is the easiest way to change the device descriptor. Let's say you want information about the printer descriptor. Enter:

OS9: xmode /p

-upc -bsb -bsl -echo -lf null=0 -pause pag=66 bsp=08
del=18 eor=0D eof=00 reprint=04 dup=01 psc=17 abort=00
quit=00 bse=5F bell=07 type=00 baud=01 xon=00 xoff=00

This print out tells about the device descriptor for the printer or /p. Minus signs tell the function is OFF and lack of a minus sign means it's ON. Otherwise numbers are included for information.

My printer needs line feeds and runs at a baud of 4800. So to change it, I enter:

OS9: xmode /p lf baud=5

This tells XMODE to change the baud rate of /p to 4800 and include line feeds with carriage returns. Simple and there is no

need to write a separate device descriptor. By the way, XMODE corrects the module's CRC so the module could be saved or used when generating a new system disk. There is one caution. XMODE only works on the descriptor and not on the path.

If a path is already open, the options must be changed in the path descriptor. To do this TMODE is used. If for example only uppercase characters are desired the following would work.

OS9: tmode upc

TMODE isshort for Terminal MODE since it is used mostoften on terminals. They are almost always open at the start of any session. XMODE is used on the device descriptors and is usually used on items like printers.

I will let you ponder these thoughts until next. Next time I will wrap this entire thought by going into the TMODE and XMODE programs. Also, I will present an interesting program called DMODE that works like XMODE, but on disk drives.

Until next time have a good month!

LISTING

```
00001          ****************************
00002          *
00003          * Name: TERM
00004          * By: Ron Voigts
00005          * Date: 16-APR-88
00006          * To compile: asm term.a o=/d0/cmds/term l $20k
00007          *
00008          ****************************
00009          *
00010          * Version 1.00  Original  ADV
00011          *
00012          ****************************
00013          *
00014          * Function:
00015          *    This is device descriptor TERM.
00016          *
00017
00018                               use   /d0/defs/defsfile
00019
00020                               ifpl
00025                               endc
00026
00027   FFA0            CPort      equ   $FFA0
00028
00029   00F1            TYPE       set   Devic+Objct
00030   0082            REVS       set   ReEnt+2
00031
00032 0000 87CD0045               mod   DDEnd,DDName,TYPE,REVS,DDMgr,DDDrv
00033
00034          * Descriptor header
00035   0000 03         fcb   $00000011  Device CapaDilities
00036   0001 07         fcb   $07        Extended Address
00037   0002 FFA0       fdb   CPort      Disk Controler Port
00038   0004 1A                 fcb   DOptEnd-*-1
00039
00040          * Initialization table
00041   0005 00           IT.DVC   fcb   $00        SCF type device
00042   0006 00           IT.UPC   fcb   $00        upper and lower case
00043   0007 00           IT.BSO   fcb   $00        backspace
00044   0008 01           IT.DLO   fcb   $01        carriage return
00045   0009 01           IT.EKO   fcb   $01        echo on
00046   000A 01           IT.ALF   fcb   $01        auto line feed on
00047   000B 00           IT.NUL   fcb   $00        end-of-line null count
00048   000C 00           IT.PAU   fcb   $00        pause off
00049   000D 18           IT.PAG   fcb   $18        lines per page
00050   000E 08           IT.BSP   fcb   $08        backspace character
00051   000F 18           IT.DEL   fcb   $18        delete line character
00052   0010 00           IT.EOR   fcb   $0D        end-of-record character
00053   0011 1B           IT.EOF   fcb   $1B        end-of-file character
00054   0012 04           IT.RPR   fcb   $04        reprint-line character
00055   0013 01           IT.DUP   fcb   $01        duplicate-last-line character
00056   0014 17           IT.PSC   fcb   $17        pause character
00057   0015 03           IT.INT   fcb   $03        interrupt character
00058   0016 05           IT.QUT   fcb   $05        quit character
00059   0017 08           IT.BSE   fcb   $08        backspace echo character
00060   0018 07           IT.OVF   fcb   $07        line-overflow character
00061   0019 80           IT.PAR   fcb   $80        initialization value
00062   001A 00           IT.BAU   fcb   $00        Baud rate
00063   001B 0037         IT.D2P   fdb   $0037      Attached device name offset
00064   001D 00           IT.XON   fcb   $00        X-ON character
00065   001E 00           IT.XOFF  fcb   $00        X-OFF character
00066   001F              DOptEnd  equ   *
00067
00068   001F 28           IT.COL   fcb   $28        number of columns
00069   0020 18           IT.ROW   fcb   $18        number of rows
00070   0021 00           IT.WND   fcb   $00        window number
00071   0022 01           IT.VAL   fcb   $01        rest of data valid
00072   0023 01           IT.STY   fcb   $01        window type
00073   0024 00           IT.CPX   fcb   $00        X position
00074   0025 00           IT.CPY   fcb   $00        Y position
00075   0026 02           IT.FGC   fcb   $02        foreground color
00076   0027 03           IT.BGC   fcb   $03        background color
00077   0028 03           IT.BDC   fcb   $03        border color
00078
00079   0029 546572ED     DDName   fcs   /Term/     Device name
00080   002D 5343C6       DDMgr    fcs   /SCF/      Device manager
00081   0030 43433349     DDDrv    fcs   /CC3IO/    Device driver
00082
00083   0034 211E70               emod
00084   0037              DDEnd    equ   *
00085                              end

00000 error(s)
00000 warning(s)
$0037 00055 program bytes generated
$0000 00000 data bytes allocated

$251E 09502 bytes used for symbols
```

+++

C User Notes

The C Programmers Reference Source. Always Right On Target!

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter continues the discussion of dbug, a C debugging package. It is a useful tool for debugging and testing C programs. It was developed by Fred Fish, who placed it into the public domain. The C code for the dbug package and for extensions to it appear in the previous, this, and in subsequent chapters.

```
DBUG.C (continued)

/*
 *      _db_pargs_      log arguments for subsequent use by _db_doprnt_()
 *
 *      VOID _db_pargs_ (_line_, keyword)
 *      int _line_;
 *      char *keyword;
 *
 *      The new universal printing macro DBUG_PRINT, which replaces
 *      all forms of the DBUG_N macros, needs two calls to runtime
 *      support routines.  The first, this function, remembers arguments
 *      that are used by the subsequent call to _db_doprnt_().
 */

VOID _db_pargs_ (_line_, keyword)
int _line_;
char *keyword;
{
    u_line = _line_;
    u_keyword = keyword;
}


/*
 *      _db_doprnt_     handle print of debug lines
 *
 *      VOID _db_doprnt_ (format, ARGLIST)
 *      char *format;
 *      long ARGLIST;
 *
 *      When invoked via one of the DBUG macros, tests the current keyword
 *      set by calling _db_pargs_() to see if that macro has been selected
 *      for processing via the debugger control string, and if so, handles
 *      printing of the arguments via the format string.  The line number
 *      of the DBUG macro in the source is found in u_line.
 *
 *      Note that the format string SHOULD NOT include a terminating
 *      newline, this is supplied automatically.
 *
 *      This runtime support routine replaces the older _db_printf_()
```

```
 *      routine which is temporarily kept around for compatibility.
 *
 *      The rather ugly argument declaration is to handle some
 *      magic with respect to the number of arguments passed
 *      via the DBUG macros.  The current maximum is 3 arguments
 *      (not including the keyword and format strings).
 *
 *      The new <varargs.h> facility is not yet common enough to
 *      convert to it quite yet...
 */

/*VARARGS1*/
VOID _db_doprnt_ (format, ARGLIST)
char *format;
long ARGLIST;
{
    if (_db_keyword_ (u_keyword))
    {
        DoPrefix (u_line);
        if (TRACING)
            Indent (stack ->level + 1);
        else
            (VOID) fprintf (_db_fp_, "%s: ", func);
        (VOID) fprintf (_db_fp_, "%s: ", u_keyword);
        (VOID) fprintf (_db_fp_, format, ARGLIST);
        (VOID) fprintf (_db_fp_, "\n");
        (VOID) fflush (_db_fp_);
        (VOID) Delay (stack ->delay);
    }
}



/*
 *      The following routine is kept around for compatibility
 *      with older objects that were compiled with the DBUG_N macro form
 *      of the print routine.  It will print a warning message on first
 *      usage.  It will go away in subsequent releases...
 */

/*VARARGS3*/
VOID _db_printf_ (_line_, keyword, format, ARGLIST)
int _line_;
char *keyword,  *format;
long ARGLIST;
{
    static BOOLEAN firsttime = TRUE;

    if (firsttime)
    {
        (VOID) fprintf (stderr, ERR_PRINTF, _db_process_, file);
        firsttime = FALSE;
    }
    _db_pargs_ (_line_, keyword);
    _db_doprnt_ (format, ARGLIST);
}



/*
```

```
 *      ListParse    parse list of modifiers in debug control string
 *
 *      LOCAL struct link *ListParse (ctlp)
 *      char *ctlp;
 *
 *      Given pointer to a comma separated list of strings in "ctlp",
 *      parses the list, building a list and returning a pointer to it.
 *      The original comma separated list is destroyed in the process of
 *      building the linked list, thus it had better be a duplicate
 *      if it is important.
 *
 *      Note that since each link is added at the head of the list,
 *      the final list will be in "reverse order", which is not
 *      significant for our usage here.
 *
 */

LOCAL struct link *ListParse (ctlp)
char *ctlp;
{
    REGISTER char * start;
    REGISTER struct link * head;
    REGISTER struct link * new;

    head = NULL;
    while (*ctlp != EOS)
    {
        start = ctlp;
        while (*ctlp != EOS && *ctlp != ',')
            ctlp++;
        if (*ctlp == ',')
            *ctlp++ = EOS;
        new = (struct link *) DbugMalloc (sizeof (struct link ));
        new ->string = StrDup (start);
        new ->next_link = head;
        head = new;
    }
    return (head);
}



/*
 *      InList    test a given string for member of a given list
 *
 *      LOCAL BOOLEAN InList (linkp, cp)
 *      struct link *linkp;
 *      char *cp;
 *
 *      Tests the string pointed to by "cp" to determine if it is in
 *      the list pointed to by "linkp".  Linkp points to the first
 *      link in the list.  If linkp is NULL then the string is treated
 *      as if it is in the list (I.E all strings are in the null list).
 *      This may seem rather strange at first but leads to the desired
 *      operation if no list is given.  The net effect is that all
 *      strings will be accepted when there is no list, and when there
 *      is a list, only those strings in the list will be accepted.
 */

LOCAL BOOLEAN InList (linkp, cp)
```

```
struct link *linkp;
char *cp;
{
    REGISTER struct link * scan;
    REGISTER BOOLEAN accept;

    if (linkp == NULL)
        accept = TRUE;
    else {
        accept = FALSE;
        for (scan = linkp; scan != NULL; scan = scan ->next_link)
        {
            if (STREQ (scan ->string, cp))
            {
                accept = TRUE;
                break;
            }
        }
    }
    return (accept);
}



/*
 *      PushState    push current state onto stack and set up new one
 *
 *      LOCAL VOID PushState ()
 *
 *      Pushes the current state on the state stack, and initializes
 *      a new state.  The only parameter inherited from the previous
 *      state is the function nesting level.  This action can be
 *      inhibited if desired, via the "r" flag.
 *
 *      The state stack ie a linked list of states, with the new
 *      state added at the head.  This allows the stack to grow
 *      to the limits of memory if necessary.
 */

LOCAL VOID PushState ()
{
    REGISTER struct state * new;

    new = (struct state *) DbugMalloc (sizeof (struct state ));
    new ->flags = 0;
    new ->delay = 0;
    new ->maxdepth = MAXDEPTH;
    if (stack != NULL)
        new ->level = stack ->level;
    else
        new ->level = 0;
    new ->out_file = stderr;
    new ->functions = NULL;
    new ->p_functions = NULL;
    new ->keywords = NULL;
    new ->processes = NULL;
    new ->next_state = stack;
    stack = new;
    init_done = TRUE;
```

```
/*
 *      DoTrace    check to see if tracing is enabled
 *
 *      LOCAL BOOLEAN DoTrace ()
 *
 *      Checks to see if tracing is enabled based on whether the
 *      user has specified tracing, the maximum trace depth has
 *      not yet been reached, the current function is selected,
 *      and the current process is selected.  Returns TRUE if
 *      tracing is enabled, FALSE otherwise.
 */

LOCAL BOOLEAN DoTrace ()
{
    REGISTER BOOLEAN trace;

    trace = FALSE;
    if (TRACING)
        if (stack ->level <= stack ->maxdepth)
            if (InList (stack ->functions, func))
                if (InList (stack ->processes, _db_process_))
                    trace = TRUE;
    return (trace);
}



/*
 *      DoProfile    check to see if profiling is current enabled
 *
 *      LOCAL BOOLEAN DoProfile ()
 *
 *      Checks to see if profiling is enabled based on whether the
 *      user has specified profiling, the maximum trace depth has
 *      not yet been reached, the current function is selected,
 *      and the current process is selected.  Returns TRUE if
 *      profiling is enabled, FALSE otherwise.
 */

LOCAL BOOLEAN DoProfile ()
{
    REGISTER BOOLEAN profile;

    profile = FALSE;
    if (PROFILING)
        if (stack ->level <= stack ->maxdepth)
            if (InList (stack ->p_functions, func))
                if (InList (stack ->processes, _db_process_))
                    profile = TRUE;
    return (profile);
}



/*
 *      _db_keyword_    test keyword for member of keyword list
 *
```

```
 *      BOOLEAN _db_keyword_ (keyword)
 *      char *keyword;
 *
 *      Test a keyword to determine if it is in the currently active
 *      keyword list.  As with the function list, a keyword is accepted
 *      if the list is null, otherwise it must match one of the list
 *      members.  When debugging is not on, no keywords are accepted.
 *      After the maximum trace level is exceeded, no keywords are
 *      accepted (this behavior subject to change).  Additionally,
 *      the current function and process must be accepted based on
 *      their respective lists.
 *
 *      Returns TRUE if keyword accepted, FALSE otherwise.
 */

BOOLEAN _db_keyword_ (keyword)
char *keyword;
{
    REGISTER BOOLEAN accept;

    if (!init_done)
        _db_push_ ("");
    accept = FALSE;
    if (DEBUGGING)
        if (stack ->level <= stack ->maxdepth)
            if (InList (stack ->functions, func))
                if (InList (stack ->keywords, keyword))
                    if (InList (stack ->processes, _db_process_))
                        accept = TRUE;
    return (accept);
}



/*
 *      Indent    indent a line to the given indentation level
 *
 *      LOCAL VOID Indent (indent)
 *      int indent;
 *
 *      Indent a line to the given level.  Note that this is
 *      a simple minded but portable implementation.
 *      There are better ways.
 *
 *      Also, the indent must be scaled by the compile time option
 *      of character positions per nesting level.
 */

LOCAL VOID Indent (indent)
int indent;
{
    REGISTER int count;
    AUTO char buffer[PRINTBUF];

    indent *= INDENT;
    for (count = 0;
        (count < (indent - INDENT)) && (count < (PRINTBUF - 1)); count++)
        if ((count % INDENT) == 0)
            buffer[count] = '|';
        else
            buffer[count] = ' ';
    buffer[count] = EOS;
    (VOID) fprintf (_db_fp_, buffer);
    (VOID) fflush (_db_fp_);
}



/*
 *      FreeList    free all memory associated with a linked list
 *
 *      LOCAL VOID FreeList (linkp)
 *      struct link *linkp;
 *
 *      Given pointer to the head of a linked list, frees all
 *      memory held by the list and the members of the list.
 */

LOCAL VOID FreeList (linkp)
struct link *linkp;
{
    REGISTER struct link * old;

    while (linkp != NULL)
    {
        old = linkp;
        linkp = linkp ->next_link;
        if (old ->string != NULL)
            free (old ->string);
        free ((char *) old);
    }
}



/*
 *      StrDup   make a duplicate of a string in new memory
 *
 *      LOCAL char *StrDup (string)
 *      char *string;
 *
 *      Given pointer to a string, allocates sufficient memory to make
 *      a duplicate copy, and copies the string to the newly allocated
 *      memory.  Failure to allocated sufficient memory is fatal.
 */


LOCAL char *StrDup (string)
char *string;
{
    REGISTER char * new;

    new = DbugMalloc (strlen (string) + 1);
    (VOID) strcpy (new, string);
    return (new);
}


/*
```

```
 *      DoPrefix    print debugger line prefix prior to indentation
 *
 *      LOCAL VOID DoPrefix (_line_)
 *      int _line_;
 *
 *      Print prefix common to all debugger output lines, prior to
 *      doing indentation if necessary.  Print such information as
 *      current process name, current source file name and line number,
 *      and current function nesting depth.
 */


LOCAL VOID DoPrefix (_line_)
int _line_;
{
    lineno++;
    if (stack ->flags & NUMBER_ON)
        (VOID) fprintf (_db_fp_, "%5d: ", lineno);
    if (stack ->flags & PROCESS_ON)
        (VOID) fprintf (_db_fp_, "%s: ", _db_process_);
    if (stack ->flags & FILE_ON)
        (VOID) fprintf (_db_fp_, "%14s: ", file);
    if (stack ->flags & LINE_ON)
        (VOID) fprintf (_db_fp_, "%5d: ", _line_);
    if (stack ->flags & DEPTH_ON)
        (VOID) fprintf (_db_fp_, "%4d: ", stack ->level);
    (VOID) fflush (_db_fp_);
}



/*
 *      OpenFile    open new output stream for debugger output
 *
 *      LOCAL VOID OpenFile (name)
 *      char *name;
 *
 *      Given name of a new file (or "-" for stdout) opens the file
 *      and sets the output stream to the new file.
 */

LOCAL VOID OpenFile (name)
char *name;
{
    REGISTER FILE * fp;
    REGISTER BOOLEAN newfile;

    if (name != NULL)
    {
        if (strcmp (name, "-") == 0)
        {
            _db_fp_ = stdout;
            stack ->out_file = _db_fp_;
        }
        else {
            if (!Writable (name))
            {
                (VOID) fprintf (_db_fp_, ERR_OPEN, _db_process_, name);
                perror ("");
                (VOID) fflush (_db_fp_);
                (VOID) Delay (stack ->delay);
            }
            else {
                if (EXISTS (name))
                    newfile = FALSE;
                else
                    newfile = TRUE;
                fp = fopen (name, "a");
                if (fp == NULL)
                {
                    (VOID) fprintf (_db_fp_, ERR_OPEN, _db_process_, name);
                    perror ("");
                    (VOID) fflush (_db_fp_);
                    (VOID) Delay (stack ->delay);
                }
                else {
                    _db_fp_ = fp;
                    stack ->out_file = fp;
                    if (newfile)
                        ChangeOwner (name);
                }
            }
        }
    }
}



/*
 *      OpenProfile    open new output stream for profiler output
 *
 *      LOCAL VOID OpenProfile (name)
 *      char *name;
 *
 *      Given name of a new file, opens the file
 *      and sets the profiler output stream to the new file.
 */

LOCAL VOID OpenProfile (name)
char *name;
{
    REGISTER BOOLEAN newfile;
    REGISTER FILE * fp;

    if (name != NULL)
    {
        if (!Writable (name))
        {
            (VOID) fprintf (_db_fp_, ERR_OPEN, _db_process_, name);
            perror ("");
            (VOID) fflush (_db_fp_);
            (VOID) Delay (stack ->delay);
        }
        else {
            if (EXISTS (name))
                newfile = FALSE;
            else
                newfile = TRUE;
            fp = fopen (name, "w");
            if (fp == NULL)
```

```
            {
                (VOID) fprintf (_db_fp_, ERR_OPEN, _db_process_, name);
                perror ("");
                (VOID) fflush (_db_fp_);
                (VOID) Delay (stack ->delay);
            }
            else {
                _db_pfp_ = fp;
                stack ->prof_file = fp;
                if (newfile)
                    ChangeOwner (name);
            }
        }
    }
}



/*
 *      CloseFile    close the debug output stream
 *
 *      LOCAL VOID CloseFile (fp)
 *      FILE *fp;
 *
 *      Closes the debug output stream unless it is standard output
 *      or standard error.
 */

LOCAL VOID CloseFile (fp)
FILE *fp;
{
    if (fp != stderr && fp != stdout)
        if (fclose (fp) == EOF)
        {
            (VOID) fprintf (stderr, ERR_CLOSE, _db_process_);
            perror ("");
            (VOID) fflush (stderr);
            (VOID) Delay (stack ->delay);
        }
}



/*
 *      DbugExit    print error message and exit
 *
 *      LOCAL VOID DbugExit (why)
 *      char *why;
 *
 *      Prints error message using current process name, the reason for
 *      aborting (typically out of memory), and exits with status 1.
 *      This should probably be changed to use a status code
 *      defined in the user's debugger include file.
 */

LOCAL VOID DbugExit (why)
char *why;
{
    (VOID) fprintf (stderr, ERR_ABORT, _db_process_, why);
    (VOID) fflush (stderr);
    (VOID) Delay (stack ->delay);
    exit (1);
}



/*
 *      DbugMalloc    allocate memory for debugger runtime support
 *
 *      LOCAL char *DbugMalloc (size)
 *      int size;
 *
 *      Allocate more memory for debugger runtime support functions.
 *      Failure to to allocate the requested number of bytes is
 *      immediately fatal to the current process.  This may be
 *      rather unfriendly behavior.  It might be better to simply
 *      print a warning message, freeze the current debugger state,
 *      and continue execution.
 */

LOCAL char *DbugMalloc (size)
int size;
{
    register char *new;

    new = malloc ((unsigned int) size);
    if (new == NULL)
        DbugExit ("out of memory");
    return (new);
}



/*
 *      This function may be eliminated when strtok is available
 *      in the runtime environment (missing from BSD4.1).
 */

LOCAL char *strtok (s1, s2)
char *s1, *s2;
{
    static ch r *end = NULL;
    REGISTER char * rtnval;

    rtnval = NULL;
    if (s2 != NULL)
    {
        if (s1 != NULL)
        {
            end = s1;
            rtnval = strtok ((char *) NULL, s2);
        }
        else if (end != NULL)
        {
            if (*end != EOS)
            {
                rtnval = end;
                while (*end != *s2 && *end != EOS)
                    end++;
                if (*end != EOS)
```

```
                *end++ = EOS;
            }
        }
    }
    return (rtnval);
}



/*
 *      BaseName    strip leading pathname components from name
 *
 *      LOCAL char *BaseName (pathname)
 *      char *pathname;
 *
 *      Given pointer to a complete pathname, locates the base file
 *      name at the end of the pathname and returns a pointer to it.
 */

LOCAL char *BaseName (pathname)
char *pathname;
{
    register char *base;

    base = strrchr (pathname, '/');
    if (base++ == NULL)
        base = pathname;
    return (base);
}



/*
 *      Writable    test to see if a pathname is writable/creatable
 *
 *      LOCAL BOOLEAN Writable (pathname)
 *      char *pathname;
 *
 *      Because the debugger might be linked in with a program that
 *      runs with the set-uid-bit (suid) set, we have to be careful
 *      about opening a user named file for debug output.  This consists
 *      of checking the file for write access with the real user id,
 *      or checking the directory where the file will be created.
 *
 *      Returns TRUE if the user would normally be allowed write or
 *      create access to the named file.  Returns FALSE otherwise.
 */

LOCAL BOOLEAN Writable (pathname)
char *pathname;
{
    REGISTER BOOLEAN granted;
#ifdef unix
    REGISTER char * lastslash;
#endif

#ifndef unix
    granted = TRUE;
#else
    granted = FALSE;
    if (EXISTS (pathname))
    {
        if (WRITABLE (pathname))
            granted = TRUE;
    }
    else
    {
        lastslash = strrchr (pathname, '/');
        if (lastslash != NULL)
            *lastslash = EOS;
        else
            pathname = ".";
        if (WRITABLE (pathname))
            granted = TRUE;
        if (lastslash != NULL)
            *lastslash = '/';
    }
#endif
    return (granted);
}



/*
 *      This function may be eliminated when strrchr is available
 *      in the runtime environment (missing from BSD4.1).
 *      Alternately, you can use rindex() on BSD systems.
 */

LOCAL char *strrchr (s, c)
char *s;
char c;
{
    REGISTER char * scan;

    for (scan = s; *scan != EOS; scan++);
    while (scan > s && *--scan != c);
    if (*scan != c)
        scan = NULL;
    return (scan);
}



/*
 *      ChangeOwner    change owner to real user for suid programs
 *
 *      LOCAL VOID ChangeOwner (pathname)
 *
 *      For unix systems, change the owner of the newly created debug
 *      file to the real owner.  This is strictly for the benefit of
 *      programs that are running with the set-user-id bit set.
 *
 *      Note that at this point, the fact that pathname represents
 *      a newly created file has already been established.  If the
 *      program that the debugger is linked to is not running with
 *      the suid bit set, then this operation is redundant (but harmless).
 */

LOCAL VOID ChangeOwner (pathname)
```

```
char *pathname;
{
#ifdef unix
    if (chown (pathname, getuid (), getgid (1) == -1)
    {
        (VOID) fprintf (stderr, ERR_CHOWN, _db_process_, pathname);
        perror ("");
        (VOID) fflush (stderr);
        (VOID) Delay (stack ->delay);
    }
#endif
}



/*
 *      _db_setjmp_     save debugger environment
 *
 *      VOID _db_setjmp_ ()
 *
 *      Invoked as part of the user's DBUG_SETJMP macro to save
 *      the debugger environment in parallel with saving the user's
 *      environment.
 */

VOID _db_setjmp_ ()
{
    jmplevel = stack ->level;
    jmpfunc = func;
    jmpfile = file;
}



/*
 *      _db_longjmp_    restore previously saved debugger environment
 *
 *      VOID _db_longjmp_ ()
 *
 *      Invoked as part of the user's DBUG_LONGJMP macro to restore
 *      the debugger environment in parallel with restoring the user's
 *      previously saved environment.
 */

VOID _db_longjmp_ ()
{
    stack ->level = jmplevel;
    if (jmpfunc)
        func = jmpfunc;
    if (jmpfile)
        file = jmpfile;
}



/*
 *      DelayArg    convert D flag argument to appropriate value
 *
 *      LOCAL int DelayArg (value)
 *      int value;
 *
 *      Converts delay argument, given in tenths of a second, to the
 *      appropriate numerical argument used by the system to delay
 *      that that many tenths of a second.  For example, on the
 *      AMIGA, there is a system call "Delay()" which takes an
 *      argument in ticks (50 per second).  On unix, the sleep
 *      command takes seconds.  Thus a value of "10", for one
 *      second of delay, gets converted to 50 on the amiga, and 1
 *      on unix.  Other systems will need to use a timing loop.
 */

LOCAL int DelayArg (value)
int value;
{
    int delayarg = 0;

#ifdef unix
    delayarg = value / 10;              /* Delay is in seconds for sleep () */
#endif
#ifdef AMIGA
    delayarg = (HZ * value) / 10;       /* Delay in ticks for Delay () */
#endif
    return (delayarg);
}



/*
 *      A dummy delay stub for systems that do not support delays.
 *      With a little work, this can be turned into a timing loop.
 */

#ifndef unix
#ifndef AMIGA
Delay ()
{
}
#endif
#endif



/*
 *      perror    perror simulation for systems that don't have it
 *
 *      LOCAL VOID perror (s)
 *      char *s;
 *
 *      Perror produces a message on the standard error stream which
 *      provides more information about the library or system error
 *      just encountered.  The argument string s is printed, followed
 *      by a ':', a blank, and then a message and a newline.
 *
 *      An undocumented feature of the unix perror is that if the string
 *      's' is a null string (NOT a NULL pointer!), then the ':' and
 *      blank are not printed.
 */

#if !unix && !(AMIGA && LATTICE)
LOCAL VOID perror (s)
char *s;
```

```
{
    if (s && *s != EOS)
        (VOID) fprintf (stderr, "%s: ", s);
    (VOID) fprintf (stderr, "<unknown system error>\n");
}
#endif  /* !unix && !(AMIGA && LATTICE) */

/*
 * Here we need the definitions of the clock routine.  Add your
 * own for whatever system that you have.
 */

#if unix

#include <sys/param.h>
#if BSD4_3 || sun

/*
 * Definition of the Clock() routine for 4.3 BSD.
 */

#include <sys/time.h>
#include <sys/resource.h>

/*
 * Returns the user time in milliseconds used by this process so far.
 */

LOCAL unsigned long Clock ()
{
    struct rusage ru;

    (VOID) getrusage (RUSAGE_SELF, &ru);
    return ((ru.ru_utime.tv_sec * 1000) + (ru.ru_utime.tv_usec / 1000));
}


#else

LOCAL unsigned long Clock ()
{
    return (0);
}


#endif

#else

#if AMIGA

struct DateStamp
{                               /* Yes, this is a hack, but doing it right */
    long ds_Days;           /* is incredibly ugly without splitting this */
    long ds_Minute;         /* off into a separate file */
    long ds_Tick;
};

static int first_clock = TRUE;
static struct DateStamp begin;
static struct DateStamp elapsed;

LOCAL unsigned long Clock ()
{
    extern VOID *AllocMem ();
    register struct DateStamp *now;
    register unsigned long millisec = 0;

    now = (struct DateStamp *) AllocMem ((long) sizeof (struct DateStamp ), 0L);
    if (now != NULL)
    {
        if (first_clock == TRUE)
        {
            first_clock = FALSE;
            (VOID) DateStamp (now);
            begin = *now;
        }
        (VOID) DateStamp (now);
        millisec = 24 * 3600 * (1000 / HZ) * (now ->ds_Days - begin.ds_Days);
        millisec += 60 * (1000 / HZ) * (now ->ds_Minute - begin.ds_Minute);
        millisec += (1000 / HZ) * (now ->ds_Tick - begin.ds_Tick);
        (VOID) FreeMem (now, (long) sizeof (struct DateStamp ));
    }
    return (millisec);
}

#endif /* AMIGA */
#endif /* unix */
```

The code for the extensions to the dbug package is contained in the next chapter.

EOF

PLus Compiler

from

Windrush Micro Systems

This is a long overdue review of the PLuS compiler from Windrush Micro Systems in England. PLuS runs under OS9 and uses OS9 system calls extensively. Let me back up and explain that I am talking about the PLuS compiler. The code generated by the compiler may be written to run under OS9 or it may be generated to run in other systems or in stand-alone hardware. One of the beauties of PLuS is that the user can define IO for himself and therefore make the object code runnable in any environment.

At the risk of repeating myself a little, I'd like to spend a few paragraphs giving you a little history of PLuS. Several years ago when the 6809 systems running FLEX were so very popular among the people who knew about them (a small but dedicated group) the PL9 compiler arrived on the scene. I, and the company for which I work, began to use PL9 exclusively for our stand alone machine control and instrumentation software. I have been using PL9 virtually every day for the past five years or more. Because of that, it is going to be difficult to write a comprehensive review, but I'll try not to leave too much out.

PLuS is a derivative of PL9, the 9 in the name of which stood for 6809. Obviously that wouldn't do for a 68008, 68000, 68010, 68020 compiler. PLuS is not claimed to mean anything in particular, but PL in the more standard mainframe language stands for Programming Language. The lower case u obviously is to represent the greek letter Mu which is commonly used as an abbreviation for "micro". We would not be far off if we said that PLuS means "Programming Language for Microcomputer Systems".

PLuS is obviously not a "standard" language. It is more or less a cross between Pascal and "C", or it could be thought of as a Pascal with some of the rules "relaxed" a little. Both Pascal and "C" have some unique syntaxes that are just fine for anyone who programs in one of them only. If you switch back and forth as I sometimes do, there are a few things that are annoying. For example Pascal distinguishes between an assignment statement and an equality test by using":=" for an assignment statement:

a:=b+3; is an assignment statement that assigns the value of b+3 to a.

a=b is the equality test.

An equality test might be part of an if-then statement. All ambiguity is removed by the difference in syntax. Similarly in "C" the distinction is made, though the difference is in the comparison and not the assignment:

a=b+3; is the "C" assignment statement.

(a==b) is the "C" comparison.

In my untrained and not-worth-a-great-deal opinion, the distinction is made clear otherwise. In the Pascal case, the comparison is always preceded by "if" or "while" or "until" which clearly indicate that what follows is a comparison. In the "C" case, comparisons are always parenthesized, though "C" has the additional perversity (in my opinion) of deliberately misinterpreting a comparison if you leave out the double equal sign. "if(a=b)..." is interpreted to mean that the programmer wants to set the value of a equal to the value of b (an assignment statement) and since the expression is clearly a comparison test requiring the return of TRUE or FALSE, it returns TRUE if a is not equal to zero and FALSE if otherwise! In "C" the not equal to zero part of a comparison is understood if not explicitly encoded if(a) is a valid comparison test that returns TRUE if a <> 0. PLuS handles the difference between assignments and comparisons the same as BASIC. It doesn't distinguish other

than by context. a=b is the assignment statement, and if a=b or while a=b or until a=b is the comparison. PLuS also takes a few of the best "shorthand" notations from "C". For example, if a then ... is a valid test meaning if a <> 0 then ... Loop counters are frequently incremented or decremented in programs. One gets tired of endlessly entering "page_count = page_count+1;" PLuS has added the increment notation of "C" so that you simply enter "page_count++;" Decrementing is done with "—". If the item being incremented or decremented happens to be a pointer into an array, PLuS knows the type of the array and increments by 1, 2, or 4 bytes.

The very latest versions of PLuS have implemented the counted loop, the for-next construct. Early versions and PL9 only had the While- and the Repeat-until loops in which a counter variable had to be set up. The latest version has all of these.

PLuS has several data types. Byte, Integer and Long are signed integers of length 1, 2, and 4 bytes respectively, their unsigned counterparts are called Char, Word, and Address respectively or alternately Ubyte, Uinteger, and Ulong if you prefer those designations. The last data type is Real, which is a four byte type that uses three bytes for the value or mantissa and one byte for the sign and exponent information. This real arithmetic has 6+ digit resolution. PLuS does a great deal of automatic type conversion if types of variables are mixed in an expression. CH = CH +32 is a valid statement though it would never pass through a Pascal program since a character would be a type CHAR and a number would be an integer. Pascal would require CH := CHR(ORD(CH)+32). That is, CH refers to the literal character. It had to be converted to the numeric representation of its ASCII value, the integer added, and the result converted back to a CHAR type. Though this process simply is a check by the compiler that the programmer knows what he is doing and it adds no code, it sometimes involves a lot of extra typing (and sometimes saves the programmer from a dumb error as well). The lack of strong typing in PL9 is a mixed blessing.

PLus doesn't get in the way when you want to add a constant to a character but it does have some automatic type conversions that can cause problems. When you have an assignment statement or a comparison PLuS converts everything in the expression to the type of the variable to which the result is to be assigned, and then does the calculation. That usually produces the desired result and there are ways to force the calculation to proceed differently. However we have been tripped up more than once by a similar conversion in a comparison. Suppose we have an integer N defined. IF N*ANGLE > PI will not work properly. N is an integer and ANGLE is therefore converted to one. Particularly if ANGLE is in RADIANS, this will be not what was wanted. The programmer has to be careful to put the REAL variable first. The comparison should be IF ANGLE*N > PI. The problem is more subtle if the comparison took the fonn IF 6<ANGLE.

The automatic type conversion also takes place when parameters are passed to a procedure. Sometimes this can be quite an advantage. For example, suppose you want to know whether the value of a variable is odd or even.

```
procedure odd(long number);
if number and 1 then return
true;
endproc;
```

That simple procedure will work for bytes, integers or longs, since all would be automatically converted when the procedure is called. Note that in PLuS the operator "and" is a bitwise function whereas ".and" is the logical function.

One of the main advantages of PLuS is that it compiles code very rapidly and quite efficiently. It is a single pass compiler and PLuS compiles the code in memory and then writes it to an output file. Since most or all of the 68008 and 68000 systems have 768K of memory minimum, this is not a limitation. Single pass compilers work by generating nulls when a forward jump or reference is found, and resolving that jump as soon as the forward referenced label is found. That is, the compiler goes back and writes over the nulls as soon as it has the correct address infonnation.

I don't have extensive information about compile times of large programs but I do have one point of reference. I translated my text editor PAT into PLuS source code. The source is just over 65K of text, and the object code is over 23K. PLuS compiles that on the 68008 system in about 30 seconds flat. I just had occasion to list the source to that, and it lists to 55 pages. That is about two pages per second.

PLuS has the facility of easily writing assembler code procedures or simply embedding a few lines of assembler in the middle of some

PLuS code. Variables can be placed at absolute addresses for such purposes as access to I/O ports in stand alone hardware or bypassing OS9 and running a device port directly. Though this is possible while running OS9, it is dangerous when running in multi-user mode. While pointers in Pascal are limited in use, those in PLuS are more like those in "C" except that only one level of indirection is allowed (you can't have pointers to arrays of pointers, that is). PLuS pointers can be great timesavers. Suppose you have an array of a dozen printer control strings each K bytes long and identified by number. You want to output the 4th string of the array called pcontrol. print(.pcontrol(4*K)) will do the job. Array references are always passed as pointers rather than passing an entire array's contents to the subprocedure. The procedure print expects a pointer to a string and it prints from that position until it finds a null. The timesaver is that you can pass a pointer to somewhere in the middle of an array.

PLuS has a very fast math package, one that is hard to improve upon with regard to execution time. The language accepts standard mathematical equations and expressions. There is a scientific function library that may be included by your program so that you can make use of sine, cosine, arctangent, square root, etc. By comparison tests we've found PLuS programs run on a 12 MHz 68008 system to run just about three times faster than the same program in PL9 on a 2 MHz 6809 system. A 10 MHz 68000 system runs about 40% faster than the 68008 system, or about five times faster than that 6809. A 68020 system runs the same program about 50% faster than the 68000 or about 7.5 times faster than the 6809.

The author of PLuS, Graham Trott, warned me early that though PL9 was written in assembler, PLuS was written essentially in itself, and that it wouldn't compile as efficiently. We were therefore pleased to find that the inefficiency of the compiled version of PLuS was much more than made up for by the faster processor. As a point of reference, PAT 6809 version compiles in PL9 in about 75 seconds compared to the 30 seconds for the PLuS version.

I would be missing an important point if I didn't mention the fact that PLuS has a built-in screen editor. When the compiler finds an error in syntax it quits and you find yourself in edit mode with the cursor right at the offending line. If you've done something dumb like leaving out a semicolon or a closing parenthesis, you can edit the line instantly and go back and try compiling again. This saves a great deal of time when compiling a long program or one translated from another language. When you exit the compiler after making changes to the program source you are asked if you want to update the source file first.

PLuS also has a debugger that allows you to run a program, stopping at any point and being able to examine the contents of variables. At compile time there are several options also. You can compile with no listing, compile with listing to a file, or to the screen. You can also have the compiled object code appear on the screen, which may be of interest to assembler programmers, and more than once has helped me find strange errors such as a forgotten close comment delimiter, which makes the compiler think the following code is just part of a long comment. A compile to the screen quickly shows that no code is generated where some should be, and the problem can be found.

An important feature of PLuS and a major improvement over PL9 is that comments are now "nestable". Previously, an end comment delimiter was taken as an absolute end of comment. For that reason you couldn't "comment out" a section of the program that contained comments. With the latest version you can. I use this feature if I am going, for example, to rewrite a procedure to try to do it better, but I want to be able to resort to the old one in case the new idea doesn't work out properly. I can comment out the old one and refer to it as I write the new one.

Another important feature when it comes to streamlining programs is the fact that the libraries are all supplied in source code. Suppose, for example, you are writing a program that only outputs information to the terminal and has no input from the user at all. You can edit IOSUBS.LIB and make yourself a special version you might call OUTPUT.LIB. You can remove over half of the original library code and minimize your program code, a feature that is particularly nice when you have just compiled a program that is three bytes too long to fit into 2 EPROMs.

Plus has only two features that I have ever considered a limitation. One is the limit of singly dimensioned arrays. You soon learn to access an array that ought to be doubly dimensioned, as for example a page of text in memory as

buffer(80*line+column). The lack is only a minor inconvenience. The other is a little harder to get around. Logical expressions are evaluated strictly from left to right and the evaluation order may not be modified by parenthesizing. That is IF A .AND B .OR C evaluates to true if A and B are true or if C is true. IF A .AND (B .OR C), which would change that sense, is not a valid construction in PL9. It is possible to get around that by using IF A THEN IF B .OR C THEN etc. Alternately you can use IF B .OR C THEN FLAG = TRUE; IF A, AND FLAG THEN ...

Another feature that can overcome the logical expression limitation is the BREAK statement. A WHILE or a REPEAT UNTIL loop can be exited from anywhere in the loop by means of an IF CONDITION THEN BREAK; statement. Break simply causes to loop to terminate and execution to start at the statement after the end of the current loop. That statement can be a test to see if the loop exited normally or because of a break condition and the appropriate action can be taken. PluS lacks one other feature found in some higher level languages. You cannot include pre-compiled modules. The .LIB files are all source code. To offset this, we have found that the compiler is so fast that it beats compilers with that feature that are usually 4 to 7 pass compilers, by a wide margin with respect to compile time anyway. Clearly a part of the reason for the speed is the fact that PLuS doesn't generate a number of intermediate code files that must be read and written to the disk.

The latest version of PLuS even contains a discussion and examples of device drivers and descriptors written in PLuS for os9. I saw a quote recently from a '68' Micro Journal subscriber that summed up his feelings about PL9 and Windrush Micro Systems in a few words. "You send your money. They send the compiler. It works." I'll add only that PLuS and PL9 are compilers that can be all things to all people. I mean of course that if you are a beginner you can ignore the libraries except to learn how to use them. If you are more advanced you can rewrite the libraries or add more of your own design to suit your own purposes including the generation of code to run on other operating systems or stand-alone. The compiler works as advertized. You'll like it.

Review by:
Ron Anderson

EOF

Logically Speaking

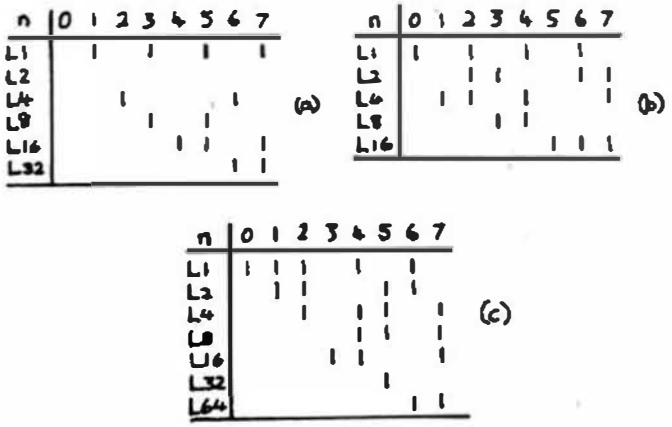
The Mathematical Design of Digital Control Circuits

Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

SOLUTIONS TO TEST TEN

1. Only the minterm tables will be given here, so you'll have to do your own decoding. Observe that 1a has no L2.

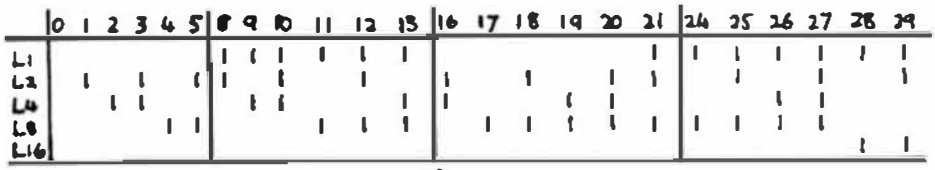


2. Again only the minterm tables will be given.

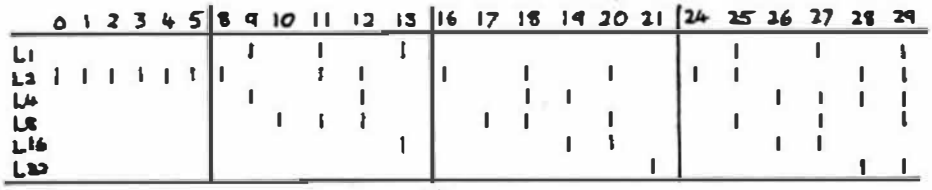
m	0						1						2						3					
n	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
	0	1	2	3	4	5	6	9	10	11	12	13	16	17	18	19	20	21	24	25	26	27	28	29
L1		1		1		1	1	1		1		1		1		1		1		1		1		1
L2			1	1			1	1			1	1			1	1			1	1			1	1
L4					1	1			1	1	1	1	1	1	1	1			1	1	1			
L8																	1	1			1	1	1	1

(d)

(b) The headings of the minterm-tables being the same for each of these three tables, I'll develop only the bottom section, and leave the rest to you. Note that because the n-imput (3-bits) is capable of counting to 7, but is only taken to 5, the 6th and 7th minterm of each m-block is omitted!

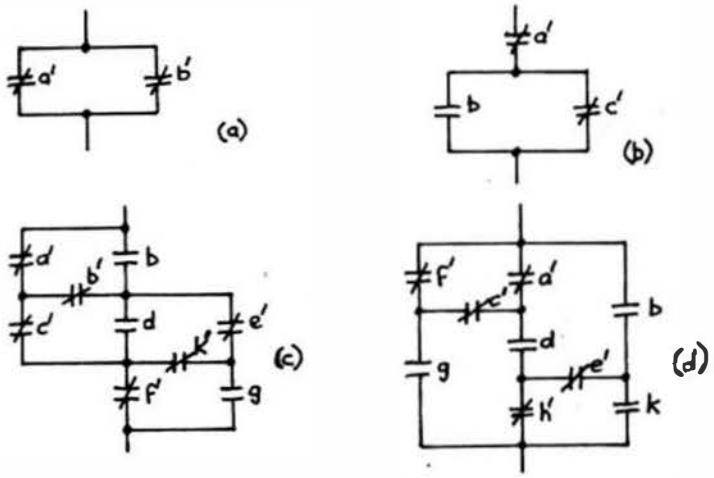


(b)



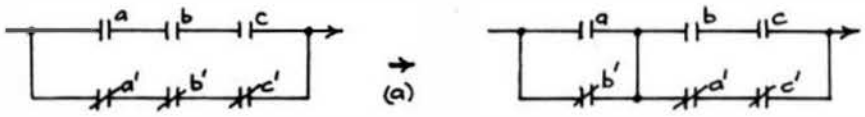
(c)

3.



As a matter of interest, the circuit of Exercise 3d will be closed (that is, transmitting) for 205 different combinations of relays, while the complemented network given in the solution above will be closed for the remaining 307 combinations.

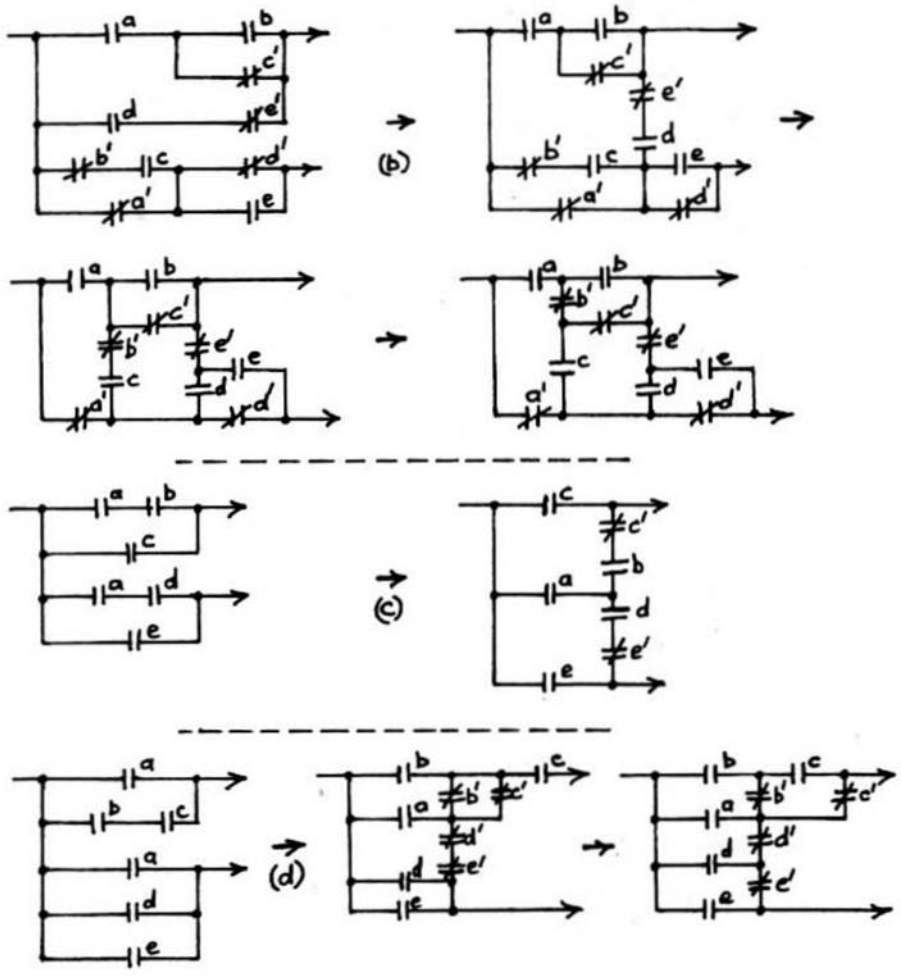
4.

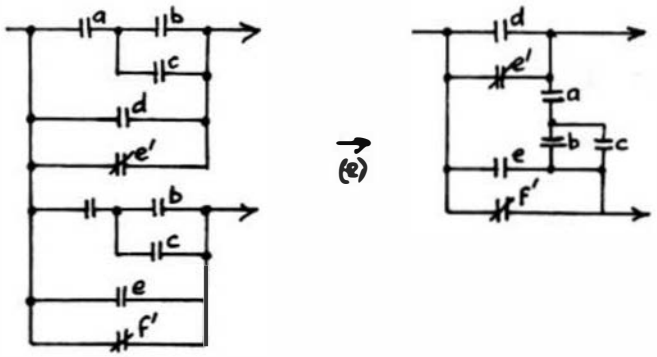


Note how transfer-contacts can be used in the right-hand network. All NO-contacts have a point in common with their corresponding NC-contact.
Algebraically, it can be shown that the two networks are equivalent

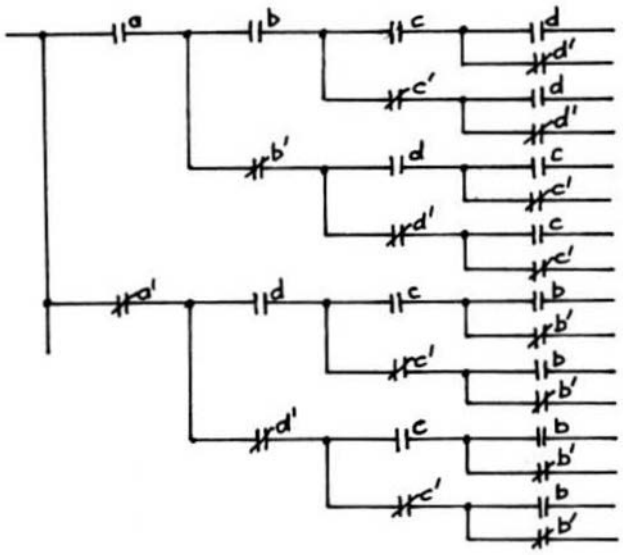
$$abc + a'b'c' = (a + b')(bc + a'c') = abc + a.a'c + b'.bc + a'b'c'$$

because both a.a'c' and b'.bc are equal to 0.

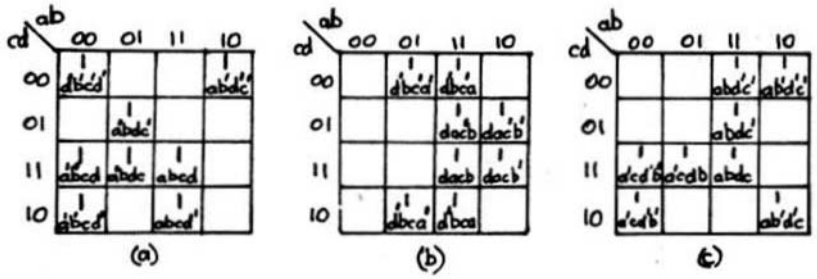




5. A B C D A B C D
1 2 4 8 equalises to 1 5 5 4



6.



Well, I guess that little lot made you stretch your mental muscles a little, but it'll do you good, as you've had it easy for quite a while! So let's all gather at

Mile 13 - heading for Mile 14.

RELAY-TREES IN 2-TERMINAL NETWORKS

Apart from multi-terminal networks (one input being directed to multiple outputs, or multiple inputs being directed to one output) relay-tree techniques can be used to design 2-terminal networks (that is, a network with one input being directed to one output). This is done by constructing one tree from the input and another from the output, and joining the appropriate ends of the two networks in the middle. As an example, consider the following expression, where f12 simply means the transmission-function from input terminal-1 to output terminal-2.

f12 = ab'c' + abc + a'bc' + a'b'c

Step one is to construct a tree for the variables "a" and "b", and an inverse-tree for variable "c". In step two we'll connect together the ends of both trees in such a way that they correctly implement the transmission function f12. Diagrams 63a and 63b show the two stages in designing the desired network.

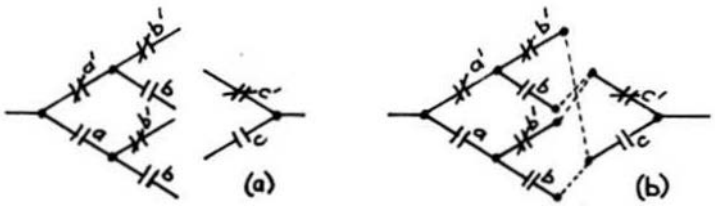


Diagram 63

It's a simple matter to read off each term of the function and to make the appropriate connection between the two trees. For example, the first term ab'c requires that the ab'-branch of the left-hand tree be joined to the c'-branch of the right-hand tree, and so on for the remaining three terms.
Generally, theorder in which the variables are divided up is important, but, unfortunately, it's mainly a matter of trial-and-error to arrive at the most economical arrangement. Of course, when dealing with more complex circuits, where both the right-hand and left-hand trees are quite large, the technique described earlier can be used to minimise each sub-tree.

ADVANCED KARNAUGH-MAPPING

READING 1S AND 0S WITH PHI

If phis appear in a set of minterms (or maxterms) which you intend to decode, whether they appear on a K-map or in decimal-minterm form, it is PARTICULARLY IMPORTANT to decode both the 1s and the 0s and then compare the resultant expressions for the better circuit. The K-map of Diagram 64 illustrates the necessity for this double decoding in order to obtain a minimum transmission function.

cd \ ab	00	01	11	10
00		1	1	
01		Ø	1	Ø
11			1	Ø
10	1	1	1	Ø

Diagram 64

Reading off the 1s, together with appropriate phis, gives a few different possibilities, such as the following, each of which, when factorised, requires five contacts for its implementation.

ab + bc' + cd' = b(a + c') + cd'
OR ac + bc' + cd' = c(a + d') + bc'
OR ad + bd' + cd' = d'(b + c) + ad

On the other hand, if we decode the 0s, again making use of the phis, we obtain the hindering function

b'c' + a'd

which, when complemented to form an alternative transmission function, results in

(b + c)(a + d')

This form requires only four relay contacts, and is obviously simpler than the one obtained by decoding the 1s. FUNCTIONALLY the two forms are identical, but ALGEBRAICALLY they're not equivalent, because in some instances a phi is read as '1' and as a '0' in the 0-decoding.

MULTI-LEVEL FACTORING

Up to this point our reading of a K-map has always resulted in a 2-level expression, that is, either a sum-of-products or a

product-of-sums expression, which we then factorised further to produce a multi-level, or "mongrel", form, as we did when we read the 1s in the previous example.

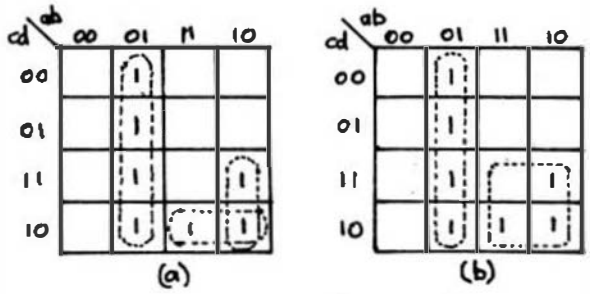


Diagram 65

The time is now ripe for you to learn how to factorise directly from the K-map, commencing with a fairly simple example, so let's look at the two K-maps of Diagram 65. 65a is looped to indicate the conventional method of reading 1s, to give the expression

a'b + ab'c + acd' = a'b + ac(b' + c')

Look now at 65b, and in particular at the loop which encloses three 1s plus one 0. If we read out both loops as though they were composed entirely of 1s, we obtain the expression

a'b + ac

which, if you compare it with the factorised expression derived from 65a, agrees with that reading, apart from the term enclosed in parens.
Now, it's clear that the term "ac" in 65b's expression is incorrect, as it includes the 0-square abcd, so we COULD make the function correct by writing

a'b + ac.(abcd)'

which we read as - a'b OR ac but NOT abcd. Here I should mention that in Boolean algebra AND and BUT are equivalent, both making use of the symbol '.' in an expression.

However, as we have the literals "ac" both inside AND outside the parens, we are permitted to delete them from the inside, so the resultant expression ends up as

a'b + ac.(bd)' = a'b + ac(b' + d')

agreeing exactly with the factorised expression of 65a.

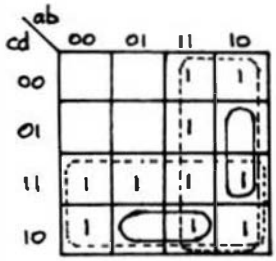


Diagram 66

With a little practice you should be able to write down the multi-level form directly from the K-map. All cases are not as simple as the one we've just done, however, so we'll develop the technique further with the map of Diagram 66. Here the two factors overlap, the two loops circled with dotted lines giving us a reading of

source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p ,u ,y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only- F, S & CCF - $49.95

Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95

OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

*Regular FLEX, SK*DOS $129.50*

** SPECIAL INTRODUCTION OFFER * $79.95*

SPECIAL PAT/JUST COMBO (w/source)

*FLEX, SK*DOS $99.95*

OS-9 68K Version $229.00

SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

*FLEX, SK*DOS $69.95*

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

*FLEX, CCF, SK*DOS $39.95*

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 - $175.00*

SPELLB "Computer Dictionary" from S.E. Media -- **OVER 150,000 words!** *Look up a word from within your Editor or Word Processor (with the SPELCMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F, S and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,

F, S or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95,

F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES 6809 CCF and CCO - $59.95,

F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!

F, S or O - $329.95, U - $549.95

O, 68000 $695.00

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems
FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character file names! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

a + c

with both terms including a 0 in their loop, which MUST BE REMOVED. Unlike our earlier example, we can here enlarge the 0-readings to include a 1, or several 1s if possible, PROVIDED THESE 1S ARE COMMON TO BOTH LOOPS. These 0-loops are shown in solid lines, from which we can see that such an arrangement is quite valid, because while we're neutralising a 1 in one loop we're leaving it untouched in the other. Thus the term "a" is transformed so

a.(ab'd)' reducing to a.(b'd)' and finally a(b + d')

while the term "c" goes through the stages

c.(bcd')' reducing to c.(bd') and finally c(b' + d)

The final combined expression is therefore a(b + d') + c(b' + d).
With this type of multiple factorisation, when 0s are coupled with 1s to form a larger elimination-loop, or inhibiting-loop, you should be VERY careful to ensure that the 1s so included also form part of another term from which they are NOT eliminated.

ENTERING PRODUCT-OF-SUMS TERMS DIRECTLY ON A K-MAP

You'll recall that somewhere near the start of our journey I told you that expressions of the form

(a + b)(c + d)

must first be multiplied out before entering them on a K-map. From now on, should the need arise, you'll enter them directly by the simple technique of mentally complementing each term in parens AND ENTERING A '0', or preferably a dot, in the appropriate location on the map. Blank squares remaining after this operation should then be filled in with 1s.
For example, the expression above will be complemented in each of its bracketed terms to give

a'b' and c'd'

and 0s entered in the column a'b' and the row c'd' of the K-map.

FIVE- AND SIX-VARIABLE KARNAUGH-MAPS

Beyond five variables, K-maps begin to get a bit unwieldy, but at the five level they're still quite useful for minimising expressions. As an example suppose we wished to minimise the following Boolean expression

abe + de + bd' + b'd + be' + bc

In order to accommodate five variables we draw two 4-variable K-maps side by side as shown in Diagram 67a, one of them being headed "e = 0" and the other "e = 1". All terms in our expression which contain a e'-variable are entered on the left-hand map only, while those which contain an e-variable are entered on the right-hand map only. Terms which don't contain the e-variable in either form are entered on both maps. By thus appearing in both maps, the e-variable effectively cancels itself out, and does not appear in the term corresponding to that entry.

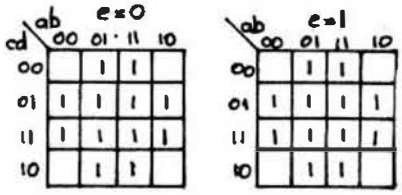


Diagram 67a

Our first step would be to enter abe on the "e = 1" map as though it were ab only, and similarly with de (entered on the "e = 1" map as though it were a simple d. Then we'd enter be' on the "e = 0" map as though it were just b. This leaves only those terms which don't contain the e-variable at all, so we enter bd' on both maps, also b'd, and finally bc. By good management on my part in making up the original expression, we end up with an identical set of 1s in both maps.
For the purpose of forming loops we must imagine the maps as being superimposed, and we're permitted to extend our loops

vertically as well as horizontally, so now we're into 3D K-maps. Thus the loop "b", being common to both maps, is an allowable loop, as is the loop "d", so our minimal expression can be read out as

b + d

If, by some chance, there happened to be an extra "1" in the top right-hand square of the "e = 1" map, we would, of course, form a loop of 4 there, and read out ac'e (don't forget the "e" on the end) and our whole expression under these circumstances would be

b + d + ac'e

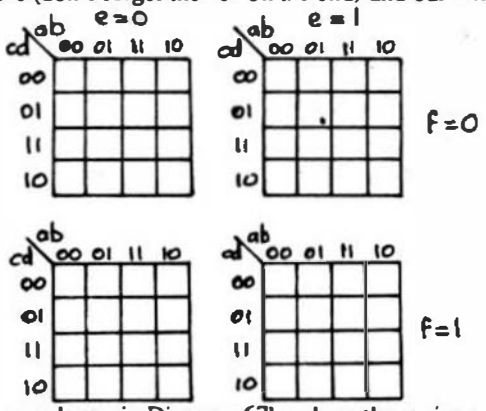


Diagram 67b

For six variables we need four K-maps as shown in Diagram 67b, where the major rows are now labelled "f = 0" and "f = 1". The rules are expanded from those of five variables, interpreting the top-left map as "ef = 00", the top-right as "ef = 10", the bottom-left as "ef = 01" and the bottom-right as "ef = 11". Thus a term containing ef' would be recorded only in the top-right K-map, a term containing no e-variable, but containing the f-variable as f', let's say, would be entered in both K-maps in the "f = 0" row, and so on. Again, the maps must be imagined as being superimposed, the major 00-map on the bottom, then the 01-map, followed by the 11-map and finally the 10-map, then the whole stack further visualised as being curled around so the top map is adjacent to the bottom one. Or, if you really want to strain your imagination, try to visualise the true situation, with the 00-map being a hollow doughnut (thus preserving all adjacencies in their correct relationship. This doughnut is then enclosed inside the 01-doughnut with corresponding squares positioned above each other, followed by an outer 11-doughnut, and finally a 10-doughnut enclosing the whole shebang. That's not TOO difficult to imagine, but you must now distort the space-time continuum so that the outer 10-doughnut is also INSIDE the innermost 00-doughnut. If you can do that successfully, you should have no problem at all in visualising a Klein-bottle!!

So, having disposed of a lot of bits and pieces over the last couple of miles, I think we're poised for a little cliff-climbing next time around, when we'll tackle a different MAJOR subject.

OK, I hear you!! Boy, you're an impatient bunch - just clamouring away for another test. Let's see if I can think of something for you! Got it!!

TEST ELEVEN

1. Design the following networks, using relay-trees

(a) f12 = a'b'c' + a'bc + ab'c' + abc

(b) f12 = a'b'c'd' + a'b'cd' + a'b'cd + a'bc'd' + ab'c'd + abc'd + abcd' + abcd

2. Draw K-maps for the following functions, and read out the multi-level factoring, marking the appropriate loops and inhibiting-loops on the maps.

The minterm-numbers must, of course, be converted to 4-bit binary before you can enter them.

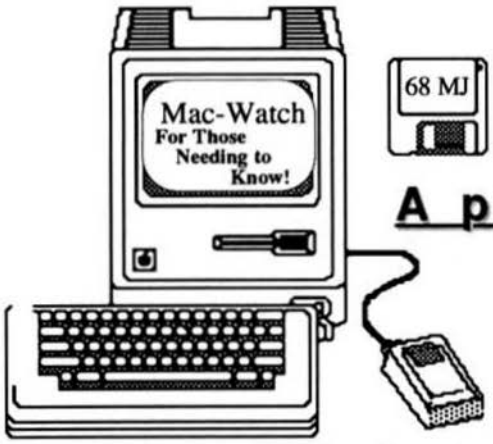
(a) f12 = 0, 2, 3, 4, 5, 6, 7, 13, 15

(b) f12 = 5, 7, 10, 11, 13, 14

(c) f12 = 0, 1, 4, 11, 12, 13, 15

(d) f12 = 2, 4, 8, 10, 14, 15 with phis = 0, 3, 5, 6, 13

... End of Mile 13. Now gathered at the Mile-14 marker!



The Macintosh™ Section

Reserved as

A place for your thoughts

And ours.....

Mac-Watch

A Review of Coach Professional

By: James E. Law
1806 Rock Bluff Rd.
Hixson, TN 37343

I never learned to be a good speller. Sometimes I blame it on early teachers who emphasized the message and down played grammar in my essays. It's probably more related to my getting in too big a hurry and not paying attention to detail. It's a good thing I have a secretary who can spell at my "real" job.

In the professional context, grammatical errors including spelling errors, are no laughing matter. The business world is too competitive to have the credibility of yourself or your company hurt by spelling errors or otherwise low quality written communications.

With the wealth of spelling checking programs available, the Macintosh user need never worry about spelling errors. One of the more full-featured such program is Coach Professional from Deneba Software. Coach Professional offers not only a wide variety of spelling checker options, but also definitions and a thesaurus for words in its comprehensive dictionary.

Professional Coach contains too many options and features to cover 100%, but I will cover enough to give you a flavor for how this program works.

What You Get...

Professional Coach is based on the 95,000 word Merriam-Webster's 9th dictionary. Also included is a 28,000 word legal and 35,000 word medical dictionary that may be merged with the main dictionary if needed. The fully merged set of dictionaries contains a whopping 158,000 words (and all that in only 343k)! For those who may be short on RAM, the Merriam-Webster's Compact 80,000 word dictionary is also enclosed. This dictionary only occupies 173k on disk. The chosen dictionary is supplemented with hyphenation file, thesaurus files, and definition files.

Professional Coach, like many post MultiFinder programs, includes both a desk accessory and a stand alone application.

Setting Up the Coach

Setting up Professional Coach is well covered by the manual and takes only a few minutes. You may have to study the manual a while, however, to really understand all the options presented in the Configuration Options box. Coach allows you to change the keyboard commands to suit your needs. You will need to be careful about conflicts with your applications, however.

Once you're set up, Professional Coach can perform batch spelling checks, perform interactive spelling checks, hyphenate words, provide definitions, and provide synonyms.

Batch Spelling Checks

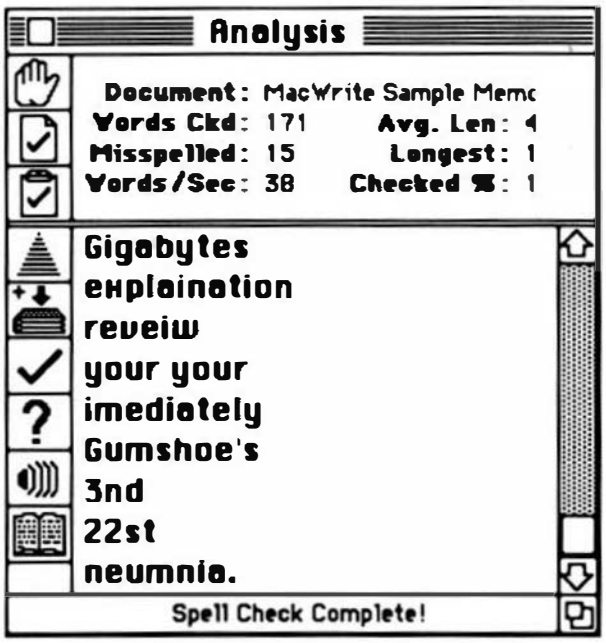
Select any text and type COMMAND + "\" to spelling check the selection. In short order Coach accumulates all potentially misspelled words in the Analysis window then immediately presents the first suspect word with suggested spellings for that word. The desired alternative may be selected through keyboard commands (e.g., COMMAND + "1") or by clicking it. This causes the misspelled word to be replaced throughout the selected text. If the same misspelling occurs several times you will only have to deal with it once. If you agree the word is misspelled but don't accept any of the alternatives as being right, you can enter command + "P" to see a list of phonetic-based (sound alike) spelling suggestions.

For example, the suspect word "neumonia" would result in "pneumonia" as a suggested correct spelling. Sometimes this option takes 30-60 seconds to complete. If even this does not result in identification of the right spelling, you can enter COMMAND + "D" to open the dictionary to the approximate position that the misspelled word might occur. You can then manually search for the right word.

Coach not only checks spelling in this mode, but also other errors such as double words, misplaced quotation marks, and capitalization errors at the beginning of sentences.

I have a number of spelling checkers and none of them present the correctly spelled word as often as Coach. In my month or soof working with this program, it was rare that the correctly spelled word was not number 1 on the list of suggestions.

If the suspect word is not in Coach's dictionary, you may **Skip** the word, **Ignore** it, or **Add** it to the dictionary. If you **Skip** the word, it will be highlighted as a suspect the next time it occurs whereas if you **Ignore** it, Coach will not bother you again with that word in the current checking exercise.



Coach Professional's Analysis Window

During the batch checking process, the Analysis window accumulates useful statistics including the number of words checked, the number of words misspelled, and the number of words checked per second. It also shows the average length of words checked and the longest word checked.

With many spelling checkers, you spend too much time waiting for a suggestion list to be developed on words you know are correctly spelled. Spelling Coach offers a very efficient way around this time waster. After spelling checking a long document with many suspects, you can stop the building of suggestion lists and select the Analysis window. This window will contain a list of all potentially misspelled words. You can go through the list cutting out all the words you know to be spelled correctly and adding others to the dictionary if appropriate. When the spelling check process continues, Spelling Coach will only spend time developing suspect lists for genuinely misspelled words. This speeds the checking up significantly for long documents with many misspelled words.

Interactive Spelling Checking

Grammar and spelling can be checked interactively by selecting **Interactive** from the Coach menu. Depending on boxes checked in the Configuration Options box, suspect words or grammar errors will result in a beep and display of typo suggestions or just a beep. When the first option was chosen, I was surprised at how quickly Coach presented a list of suggestions after an apparent error (about one second, most of the time). As I stated before, this list almost always contains the correct pronunciation or spelling, usually as the first item in the list. A click on the desired selection or a keyboard command causes the error to be quickly corrected. Just as with batch checking, you can ask for phonetic suggestions, open the dictionary, add the word to the dictionary, skip the word, or ignore it.

What Does It Mean?

If all Coach Professional did was check spelling, it would be worth owning, but it does even more to help you make your writing as good as it can be. One important function is to present the definition of any word in its dictionary. The definition window is obtained by selecting a word and typing COMMAND + "7". The window then shows alternate definitions, the spelling of various forms of the word, and the part of speech of each (e.g., verb, adjective, noun). The window scrolls vertically, as needed, to show hidden text.

This function is very versatile. The word to be defined can be chosen from the document being checked or from a Coach Professional window.

Saying It Another Way

Professional Coach displays a complete set of synonyms for each major meaning of words in its dictionary. For example, 11 sets of synonyms are presented for the word "just" with a total of 80 entries. This information can be presented in a choice of 2 formats. The "List Format" displays a small window that scrolls vertically to reveal more synonyms in a set and scrolls horizontally to reveal different sets of synonyms. The "Text Format" squeezes all groups of synonyms into a format where the most possible information is viewable at the same time. Which one you choose will be a matter of personal preference.

Working Together

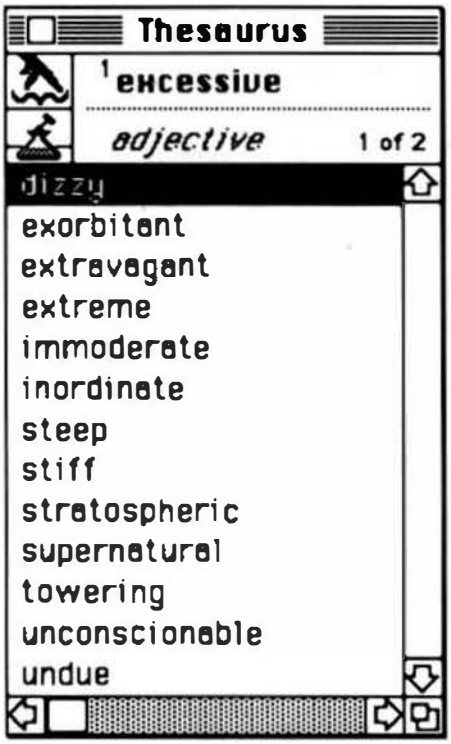
Coach Professional's features are nicely integrated so that they compliment each other. For example, you can select a word from your document look up its definition to ensure it is the right word. At the same time, you can view another window showing synonyms for the word. You can then select words from either of these windows and view the Definitions or Thesaurus window for these new words.

Should You Buy Coach Professional?

Coach Professional works with a 512E, MacPlus, SE, or II. As a minimum, two disk drives are required. A hard disk is essential, however, to get the most from this program's many features.

I thought Coach Professional was quite complicated when I first looked at it. The manual's comprehensive coverage of the many options led me to this conclusion. After seriously working with Coach Professional for a while, I changed my mind. Memorizing less than 5 keyboard commands and the meaning of a few icons will let you use most of its features.

Coach Professional is the best spelling checker that I have used to-date. If it was just a spelling checker, it would be a quality product. The additional definitions and thesaurus features make it a solid value. I recommend this product.



Tresaurus Window

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01453

MORE ON STEARNS' COLOR-FORTH

I have been able to get my copy of COLOR-FORTH to run on my CoCo3 with a minimum number of compromises; so I thought that I would report on my progress. I have mentioned COLOR-FORTH in previous columns, including a fairly detailed review, but that was for a CoCo1.

The main problem, which I have not yet solved, is getting the screen-oriented editor to function properly on the CoCo3. Whenever I try to load a screen for editing with the command

I-1105 E <return>

for loading screen #105, the system locks up and the display goes wild. This forces me to reset the CoCo3 to a COLD START! Needless to say, I would prefer a different result.

Temporarily, I sidestepped the problem by not using this editor at all. Instead, I used the primitive P command to write directly to a screen. This was not too bad, since there are only 32 characters per line, so there was not so much there to be corrected if an error showed up. My only real complaint was that there is no obvious way to spread lines for inserting a modification to an existing definition. The only way to do this was to retype the whole screen, beginning at the desired change.

The sequence of operations I used with the P command are listed in Figure 1.

I have gotten around most of the editing problems by adapting the FORTH line editor supplied with FF9. This editor was originally written by S H Daniel and appeared in FORTH DIMENSIONS, vol III, no. 3. Actually, I just copied the listing from FORTH DIMENSIONS, making the few changes necessitated by the slight difference in the available words (ENDIF for THEN , etc.). This has worked out fine, and I may quit while I am ahead.

The line editor, as I now use it for COLOR-FORTH, is shown in screens #30 - 39. I will not try to explain in detail how the editor works; just take my word for it. I know that the listing is hard to read in this compressed format, but I did it this way to save space on the disk. Please refer to Brodie's "Starting FORTH" for instructions on using this editor. It functions identically to the one described in the first edition, and is the line editor described in the second edition. This editor may be used with any display.

Screen #30 is only the the loading screen; you load the editor by commanding

I-130 LOAD <enter>

I must emphasize that disk I/O only works with 512-byte screens, so you must adapt your programming style to fit within this limitation. Actually, this is not such a big deal; see the explanation below. I find that I can get along quite well with the standard screen format provided with COLOR-FORTH, so any changes are way down on my list of priorities.

Figure 1. The steps in editing a Stearns' COLOR-FORTH screen without using the "S8 Editor."

1. (screen number) LIST <return>
2. Press any key to return to command mode.
3. (line number) P <return>
4. Type the required text, limited to 32 characters. <return>
5. Repeat steps 3 and 4 as many times as necessary, limited to line numbers 0 - 15.
6. L <return> to review the screen.
7. Press any key to return to command mode.
8. FLUSH <return> to save the current screen.

Using the 80-column Display

Much higher on my priority list was getting the 80-column "hardware" display within the CoCo3 to respond to COLOR-FORTH. I have done so, and am quite happy with the result. In fact, I am now using the 80x28 display. This comes up from a cold start using the BASIC program in Figure 2. Most of the information utilized in writing this boot came from information downloaded from CompuServe; I must abjectly apologize for not being able to identify the authors of the various parts I used, but I have lost the appropriate references; anybody who can enlighten me please do so and I will give proper credit.

```
10 RGB:WIDTH80:CLS3:ATTR3,2 : ' Initialize to 80-col, white on blue
20 POKE &H95C9,&H74:POKE &HFF22,&H10 : ' Lower case on 32x16
30 POKE &HFF40,0  : ' Turn off disk drive
40 CW=PEEK(&HE7)  : ' Current screen type, so can restore it afterwards.
50 WIDTH 32       : ' For some reason, this is necessary ????
60 READ A,V
70 IF A=0 THEN WIDTH (16*CW-8)*CW+32
   :PRINT"28 line text screens now enabled." : GOTO220
80 POKE A,V
90 GOTO 60
100 DATA &HE046,&H75
110 DATA &HE03D,&H65
120 DATA &HF688,&H31 ,  &HF689, &H80
130 DATA &HF66B,&H28 ,  &HF66C, &HC0
140 DATA &HF875,&H30 ,  &HF876, &HE0
150 DATA &HF866,&H28 ,  &HF867, &H70
160 DATA &HF683,&H1C
170 DATA &HF666,&H1C
180 DATA &HF87F,&H1B
190 DATA &HF8F4,&H1C
200 DATA &HF69D,&H8C , &HF69E,&H32 , &HF69F , &H40
210 DATA 0,0
220 VERIFYON
230 REM 11/03/87, RDL
240 CLS
250 PRINT "Place FORTH disk into drive #0"
260 PRINT
270 PRINT "Press any key to load FORTH"
280 PRINT
290 POKE &HFE08,&H9A : ' Start blink
300 PRINT "Ignore the OS error message."
310 POKE &HFE08,&H1A : ' Stop blink
320 PRINT
330 PRINT "Type EXEC 1536 to run FORTH":PRINT:PRINT
340 A$=INKEY$:IF A$="" THEN 340
350 DSKI$ 0,0,1,A$,A$
```

Figure 2. *The BASIC bootstrap program for COLOR-FORTH on the CoCo3.*

All I need to do to load COLOR-FORTH is to execute this BASIC program and then follow the directions on the screen. Once the additional screens, starting at 20 have been loaded, I type the following command:

I-IW32 W80 <ENTER>

Screen #27 has the necessary definitions for moving back and forth between the 80x28 and 32x16 displays. I don't know why, but it is necessary to call up the 32x16 display momentarily in order to get the 80x28 screen to do a proper carriage return/line feed. Sometimes, the 80x28 screen gets fouled up during use and I have to use the W32 W80 command line to fix the problem. I have no idea why this happens, but it is so easy to fix that it has gotten pushed way down on my priority list.

Shadow Screens

I have found that the only significant limitation to the 32x16 screen in COLOR- FORTH is that it leaves virtually no room for comments. Normally, I like to put in a lot of comments so that I can tell at a glance just what a definition is expected to accomplish and how it goes about it. However, a line only 32 characters long puts a severe crimp in my style of programming; therefore, I decided to make a virtue out of necessity.

I am sure that practically all of you have heard of the concept of "shadow screens" for comments. That is the technique that I have decided to use with COLOR-FORTH. I dawned on me that disks are now so cheap that there is no reason to worry about squeezing every last word onto a minimum number of lines, so I can be pretty free about "wasting" space. I have double-sided drives with my CoCo3 which are set up to use the back side of drive #0 as drive #2 and the back side of drive #1 as drive #3. By using the odd-numbered drives as the shadow, I could have a 32x16 screen for programming and a 32x16 screen for comments. Furthermore, the 80x28 display was perfect for showing the two side by side, pretty much as a 64x16 FORTH screen, only better. The listings for screens 28-29 show what I mean.

I am working on some definitions for the line editor which will make it easy to edit the shadow screens. I'll publish them as soon as they are ready.

Color Demonstration

I have also included definitions for RGB and CMP in SCR #27, as they may be needed by some people. I also use these two definitions to make very quick changes in some of the colors, as they look different on my CM-8 rgb monitor.

Screen #50 has a simple definition which I find amusing. It displays all of the color combinations of letters and background normally available by changing the attribute byte of the display descriptor. There is a visually instantaneous color change with this display when going from RGB to CMP, and vice versa.

Frankly, now that I have gotten COLOR-FORTH to work on my CoCo3, I am even more impressed with it than I was before; but I still have to admit that I prefer FF9. I just have to get FF9 to work with 80 columns. Actually, what I prefer is the FORTH-83, so I may try to convert COLOR-FORTH, instead. (For those who are interested and may not have noticed, COLOR-FORTH is available from CP1.)

THE ULTIMATE IN HARDCOPY

At the last local FIG meeting, Dave Lindbergh (a consultant) and Dick Miller (Miller Microcomputer Services) described what must be the ultimate in shrinking hardcopy. By using a laser printer at 300 by 300 dots per inch, they were able to produce FORTH screens the size of a postage stamp; these screens could still be read by the use of a very strong magnifying glass. They were able to use a 24-pin dot matrix printer to print 3 lines where I would normally be printed; this resulted in 27 screens per printed page which could be read without any magnification. The entire program for either printing method takes 9 screens, and the font description takes additional screens, the number depending on the dot-pattern chosen. A 7x5 pattern has 6 characters per screen. I don't have a 24-pin printer, so I have not done any experimenting with this application, yet, but I will report if I find it useful other places. If you are interested in more details before I get around to a column report, just contact me or Dick Miller; this is a public domain program.

FEEDBACK

I got an interesting letter from Paul Pallmer of Pasco, WA, commenting on my April column. He felt that my method of calculating sigma was dangerous, since there was the possibility of trying to work with small differences between very large numbers. In theory, I agree completely, but in about 20 years of using the exact technique I described, I have never had a real engineering or scientific problem affected by this potential flaw in the algorithm. When one deals with only 4 or 5 significant figures, any differences are usually handled easily by the math hardware/software. I still think that I was justified in my choice of technique, but others should be warned that there is potential for grave error.

Let me digress for a moment to "ride a hobby-horse" of mine. I have had long discussions/arguments with other engineers over the question of cross-sectional area of a test specimen. As a specific example, consider a tensile test bar which a technician has measured to be 0.497 inches wide by 0.121 inches thick. She used a micrometer callibrated to 0.001 inches, so we assumed that she rounded the last digit appropriately. Now the question is whether the area is 0.060137 square inches or 0.060 square inches. I have always maintained that the latter must be true, since the potential real value could range between 0.496 * 0.120 = 0.059520 and 0.498 * 0.122 = 0.060756. Any way, I think that too many people put too much blind faith in numbers, without thinking about where those numbers came from. I can't help wondering if some of NASA's problems don't arise from a similar blind faith in a long list of insignificant digits.

TO FACTOR OR NOT TO FACTOR, THAT IS THE QUESTION.

Paul also brought up in his letter that I could have factored my definition of SIGMA much more appropriately. I have to agree, again, but....

I have a strong faith in the good points of factoring definitions. The editor shown in screens 30 - 38 is an example of good factoring. Usually, a properly factored application is self-documenting, because the names of the primitives are formed from the pseudo-equations used to generate the algorithm. In fact, if it is difficult to think of a suitable name for a particular primitive, then that is the wrong primitive. Exceptions do exist, but that is begging the issue.

On the other hand, there are times when you don't want to factor a definition. Usually, this comes from a definition that is too simple and obvious to bother with, as in screen #27 in the definitions of W80 and W32 . There is a phrase,

```
HE7 C! F67D JSR
```

which is common to both definitions. It could be called CHANGE-WIDTH and preceeded by either 0 or 2, but that would confuse more than clarify the definition.

However, there is another consideration. A factored definition does take longer to execute than the same unfactored definition. This is because you must pass through NEXT every time you move from factor to factor, and this always has an unavoidable overhead. Rarely is this an important factor, but it should be considered; the few microseconds saved may keep you from having to write a definition in assembly language.

As a final point, don't factor a working definition just because it would look neater that way! Since I never expect to be using any of the potential factors of SIGMA from my standard deviation calculation any where else, I decided to quit while I was ahead. The application worked, so don't mess with it!

MORE ON FEEDBACK

Paul had some other comments which I want to comment on later, but that is enough for now. But this does bring up another point, I can't cover what you want to see if you don't tell me about it. Let me know what you want, and I will try to cover it. Even though I am not a real FORTH expert, I do have access to some people who are the leading lights in the FORTH community, and I can relay your questions to them.

Listing:

```
SCR #  27
  0 \ W80 W32 RGB CMP       RDL042588<
  1     HEX                          <
  2 : W80    ( - )        \ RDL042488<
  3     2 E7 C!    F67D JSR ;        <
  4                                  <
  5 : W32    ( - )        \ RDL042488<
  6     74 95C9 C!  10 FF22 C!       <
  7      0   E7 C!  F67D JSR ;       <
  8                                  <
  9 : CMP    ( - )        \ RDL042588<
 10     E676 JSR ;                   <
 11                                  <
 12 : RGB    ( - )        \ RDL042588<
 13     E674 JSR ;                   <
 14                                  <
 15 DECIMAL  SIN  ;S                 <

SCR #28
 0 \ LS                    RDL051688 \ LS
RDL051688
 1 FORTH DEFINITIONS DECIMAL        \ Display SCR# &
SHADOW on 80 col.
 2 630 CONSTANT SHADOW              \ Offset to back of
disk
 3 : LS    ( scr# - )               \
 4    W80                           \ Force 80-column
screen
 5    CR ." SCR #" DUP .            \ Print screen
number
 6    16 0 DO                       \ 16 lines
 7      CR I 2 .R  SPACE            \ Print line number
 8      DUP BLOCK 32 I * +          \ Duplicate scr# &
point to
 9       32 TYPE  ." \ "            \  line, print it,
print delim
10      DUP SHADOW + BLOCK 32 I *   \ Duplicate scr# &
point to
11       +  32 TYPE                 \  shadow line;
print it
12    LOOP                          \
13    SCR !  CR ;                   \ Update SCR
14 SIN ->                          \
15                                  \

SCR #29
 0 \ >L <L                 RDL051688 \ >L <L
RDL051688
 1 : >L    ( - )                   \ display next
screen
 2     1 SCR +!                     \ increment the
value in SCR
 3     SCR @ LS ;                   \ display the new
screen & shadow
 4                                  \
 5 : <L    ( - )                   \ display previous
screen
 6     -1 SCR +!                    \ decrement the
value in SCR
 7     SCR @ LS ;                   \ display the new
screen & shadow
 8                                  \
 9 SIN ->                           \

SCR # 30
  0 \ LINE EDITOR          RDL042788 <
  1 \ Loading Screen                 <
  2                                  <
  3 31                               <
  4 39                               <
  5 THRU                             <
  6                                  <
 15 DECIMAL  SIN ;S                  <

SCR # 31
  0 : (MATCH) -DUP IF OVER + SWAP DO<
  1  DUP C@ I C@ - IF 0= LEAVE ELSE <
  2 1+ ENDIF LOOP ELSE DROP 0=       <
  3 ENDIF ;                          <
  4 : MATCH >R >R 2DUP R> R> 2SWAP   <
  5 OVER + SWAP DO 2DUP I SWAP       <
  6 (MATCH) IF >R 2DROP R> - I SWAP  <
  7 - 0 SWAP 0 0 LEAVE ENDIF LOOP    <
  8 2DROP SWAP 0= SWAP ;             <
  9 32 CONSTANT C/L                  <
 10  1 CONSTANT B/SCR                <
 15 SIN  ;S                          <

SCR # 32
  0 FORTH DEFINITIONS HEX            <
  1 : TEXT HERE C/L 1+ BLANKS WORD   <
  2 HERE PAD C/L 1+ MOVE ;           <
  4 : LINE DUP FFF0 AND IF           <
  5 ." NOT ON CURRENT EDITING SCREEN<
  6 " ABORT ENDIF SCR @ (LINE)       <
  7 DROP ;                           <
  9 SIN ;S

SCR # 33
  0 VOCABULARY EDITOR IMMEDIATE HEX <
  1 : WHERE DUP B/SCR / SCR !        <
  2 ." SCR # " DECIMAL . SWAP C/L    <
  3 /MOD C/L * ROT BLOCK + CR C/L    <
  4 TYPE CR HERE CR C@ - SPACES 5E   <
  5 EMIT [COMPILE] EDITOR QUIT ;     <
  6 EDITOR DEFINITIONS               <
  7 : #LOCATE R# @ C/L /MOD ;        <
  8 : #LEAD #LOCATE LINE SWAP ;      <
  9 : #LAG #LEAD DUP >R + C/L R> - ;<
 10  : -MOVE LINE C/L MOVE UPDATE ; <
 11 : BUF-MOVE PAD 1+ C@ IF PAD SWAP<
 12  C/L 1+ MOVE ELSE DROP ENDIF ;  <
 13 : >LINE# #LOCATE SWAP DROP ;    <
 14 : FIND-BUF PAD 50 + ;           <
 15 SIN ;S                          <

SCR # 34
  0 : INSERT-BUF FIND-BUF 50 + ;     <
  1 : (HOLD) LINE INSERT-BUF 1+ C/L <
  2 DUP INSERT-BUF C! MOVE ;         <
  3 : (KILL) LINE C/L BLANKS         <
  4 UPDATE ;                         <
  5 : (SPREAD) >LINE# DUP 1 - 0E DO <
  6 I LINE I 1+ -MOVE -1 +LOOP       <
  7 (KILL) ;                         <
  8 : X >LINE# DUP (HOLD) 0F DUP ROT<
  9  DO I 1+ LINE I -MOVE LOOP       <
 10 (KILL) ;                         <
 11 : DISPLAY-CURSOR CR SPACE #LEAD <
 12 TYPE 5E EMIT #LAG TYPE #LOCATE  <
```

```
 13 . DROP ;                          <
 15 SIN  ;S                           <

SCR # 35
  0 : T C/L * R# ! 0                 <
  1 DISPLAY-CURSOR ;                 <
  2 : (TOP) 0 R# ! ;                 <
  3 : SEEK-ERROR (TOP) FIND-BUF HERE<
  4  C/L 1+ MOVE HERE COUNT TYPE     <
  5 ." NONE" QUIT ;                  <
  6 : (R) >LINE# INSERT-BUF 1+ SWAP  <
  7 -MOVE ;                          <
  8 : P 5E TEXT INSERT-BUF BUF-MOVE  <
  9 (R) ;                            <
 11 : L SCR @ LIST ;                 <
 15 SIN  ;S                          <

SCR # 36
  0 : COPY B/SCR * OFFSET @ + SWAP   <
  1 B/SCR * B/SCR OVER + SWAP DO DUP<
  2  FORTH I BLOCK 2 - ! 1+ UPDATE   <
  3 LOOP DROP FLUSH ;                <
  4 : 1LINE #LAG FIND-BUF COUNT      <
  5 MATCH R# +! ;                    <
  6 : (SEEK) BEGIN 3FF R# @ < IF     <
  7 SEEK-ERROR ENDIF 1LINE UNTIL ;   <
  8 : (DELETE) >R #LAG + R - #LAG R  <
  9 MINUS R# +! #LEAD + SWAP MOVE    <
 10 R> BLANKS UPDATE ;               <
 11 : (F) 5E TEXT FIND-BUF BUF-MOVE  <
 12 (SEEK) ;                         <
 13 : F (F) DISPLAY-CURSOR ;         <
 15 SIN  ;S                          <

SCR # 37
  0 : (E) FIND-BUF C@ (DELETE) ;     <
  1 : E (E) DISPLAY-CURSOR ;         <
  2 : D (F) E ;                      <
  3 : TILL #LEAD + 5E TEXT FIND-BUF  <
  4 BUF-MOVE 1LINE 0= IF SEEK-ERROR  <
  5 ENDIF #LEAD + SWAP - (DELETE)    <
  6 DISPLAY-CURSOR ;                 <
  8 0 VARIABLE COUNTER               <
  9 : BUMP 1 COUNTER +! COUNTER @    <
 10 30 > IF 0 COUNTER ! CR CR 0F     <
 11 MESSAGE 0C EMIT ENDIF ;          <
 15 SIN  ;S                          <

SCR # 38
  0 : S C EMIT 5E TEXT @ COUNTER !   <
  1 FIND-BUF BUF-MOVE SCR @ DUP >R   <
  2 DO I SCR ! (TOP) BEGIN 1LINE IF  <
  3 DISPLAY-CURSOR SCR ? BUMP ENDIF  <
  4 3FF R# @ < UNTIL LOOP R> SCR ! ;<
  6 : I 5E TEXT INSERT-BUF BUF-MOVE  <
  7 INSERT-BUF COUNT #LAG ROT OVER   <
  8 MIN >R R R# +! R - >R DUP HERE   <
  9 R MOVE HERE #LEAD + R> MOVE R>   <
 10 MOVE UPDATE DISPLAY-CURSOR ;     <
 12 : U C/L R# +! (SPREAD) P ;       <
 13 : R (E) I ;                      <
 15 SIN  ;S                          <

SCR #  50
  0 \ #0DEMO1                  04/24/88<
  1 HEX                                <
  2 FE08 CONSTANT ATTR                 <
  3                                    <
  4 : #0DEMO1      ( - )   \ 04/22/88<
  5     CR  40 0 DO                    <
  6       I ATTR C!                    <
  7       I 28 .R                      <
  8     LOOP                           <
  9     1A ATTR C! ; \ restore color<
 10 DECIMAL    SIN    ;S              <

EOF
```

Bring your FLEX Disk system to todays technology

HIER Enhancements and Extra Utilities

Dave Parr
Computer Science Dept.
Coventry Polytechnic, Priory Street
Coventry, England CV1 5FB Tel: (203) 838710

Some time ago I purchased HEIR from S.E. Media, having been impressed by the specification given in the advertisement. First of all let me state that it is a very clever piece of software and does everything claimed; however I have to say that I soon came upon a problem with my particular system configuration, and since there could be a number of people in a similar situation, my solution may be of interest.

I wanted to use HIER on a FLEX system which has a 10mb.winchester and a single 5" floppy drive. While I assigned the winchester to be the system drive and the floppy to be the work drive, I could create and move into sub-directories with no problem at all. However, what I really wanted was both system and work drive assigned to the winchester, because that is where I was going to be storing a large number of files.

When I created sub-directories on the winchester and 'changed-directory' into one of them, I found that normal FLEX system commands were no longer accessible unless I either called them via the RUN utility supplied with HIER, or copied the commands from the HOME directory into each and every sub-directory. Obviously neither of these alternatives was satisfactory, so I set about producing some modifications to HIER which would solve the problem.

What I have come up with is a number of lines of code to be inserted into HIER, and two extra utility programs in addition to those supplied. I have discussed this matter with Ray Goff, the author of HIER, and he is happy for me to give details of my modifications in such a way as to be useful to people who have purchased HIER, but obviously without giving away secrets of his product. Therefore I will give an explanation of my modifications, followed by details of how to add these to the HIER source code. N.B. These modifications apply only to the FLEX version of HIER, not to the SK*DOS version.

THEORY

The trick is to hijack FLEX at the point where it opens a command file for reading (and subsequent loading and execution). Normally HIER forces the directory search for a file to be made in the current directory, so HIER must be informed that in this case, the search should be made in the HOME directory. To achieve this, a patch is inserted in FLEX (before it opens the command file for reading) which causes a jump to a few extra lines of code added to HIER. This code sets a flag byte to indicate to HIER that the open-for-read search is to be performed in the HOME directory. After the command file has been successfully opened, the flag byte is cleared again so that HIER will look in the current directory for any filenames which are arguments on the command line, or for workfiles. The code then jumps back to FLEX where it left off.

User commands in the current directory can still be invoked by means of the RUN utility. Also, if a command is to be executed from a hierarchically-structured disk in the work drive, the command will be searched for in the HOME directory of that disk; therefore if the command is in a different directory, the RUN utility should again be used. Since HIER relocates itself and alters the end-of-memory when it is first run, the extra instructions added to it are allowed for automatically. Changing HIER in this way involves no disturbance to an existing system, assuming that all system commands have been placed in the HOME directory; it is simply a matter of re-assembling HIER and invoking the new version at STARTUP. However, there is another possibility, so read on...

Personally I would rather reserve the HOME directory as much as possible for sub-directories, placing all system commands at a lower level in a CMD sub-directory. Fortunately, making this work is also very easy, simply by adding two extra instructions to the modifications, but it does mean that the system disk has to be re-organised somewhat so that the command files are in the right place. There are many ways to achieve this, for instance I found it simple to work from a system floppy disk in my 5" drive, running the original version of HIER, until I had correctly organised the winchester, then re-booting and running the new version of HIER to take future commands from the CMD sub-directory. If you are prepared to re-format your winchester, you can create CMD as the first sub-directory on the disk, which should keep access times down.

The one remaining problem with HIER which is not solved by these modifications is that of library files, include files and files like PRINT.SYS and ERRORS.SYS, which also cannot be found unless they exist in the current working directory. My solution to this was to provide links in the current directory to files in other directories which might be needed; I do this by means of a link program LN which I have written. The link entries can, if desired, be catalog-protected to cut down on output when listing the contents of a directory. I have also written an unlink program ULN to safely remove unwanted link entries from a directory, since the FLEX command DELETE cannot be used.

I show below my modifications to HIER, but the LN and ULN programs (written in assembler) are too long to include here. However I am prepared to supply these in executable form on disk, at a small charge to cover the cost of media, shipping and handling. I don't think I should supply sources since these contain some sections of code which parse the directory paths in a similar manner to a number of the utilities sold with HIER. If anybody is interested, please let me know disk size (8, 5.25 or 3.5") and the type of formatting required, and enclose a check/cheque for $15 U.S. or #8 Sterling, made payable to "Coventry Polytechnic".

ADDING MODIFICATIONS

Add to HIER.TXT, between lines 257 and 258, the following:

```
DSKCM1 LDA #$02
       STA COMFLG,PCR
       JSR OPNFRD
       CLR COMFLG,PCR
       JMP DSKCMR
COMFLG FCB 0
```

Next add, between lines 172 and 173, the following:

```
       TST COMFLG,PCR
       BEQ RDSIR
       LDD #CMDDIR
       BRA OK
RDSIR  EQU *
```

Now add, between lines 153 and 154, the following:

```
       STA DSKCMD
       LEAX DSKCM1,PCR
       STX DSKCMD+1
       LEAX HIER,PCR
```

Finally, it is necessary to define the extra labels used in these inserted lines. Ideally this would be a job for an installation program, but for the purposes of this article I will describe how to find the necessary values.

Firstly, the labels DSKCMD and DSKCMR must be located in FLEX; in most versions these will be at $D22E and $D232 respectively, but you may possibly have a version of FLEX where this differs slightly. In that case, the correct locations can be identified by searching in the same area for the string of bytes 86 02 8D 22 8D EA BD D1 A2 The location of the first byte (86) is the value for the label DSKCMD and the location of the fifth byte (8D) is the value for DSKCMR. Similarly, the label OPNFRD should be located a little further on, probably at location $D254. The string of bytes to look for in order to check this is 8E C8 40 BD D0 EB 8E C8 40 the value for the label being the location of the first 8E. If you are unable to find either of these strings of bytes, it is not advisable to proceed further, unless you feel competent to investigate FLEX more deeply.

Finally the label CMDDIR, which is actually the track/sector address where the directory which holds all the system commands begins on the winchester. For the first set of modifications described, this will be the HOME directory which begins at track 0 sector 5, therefore the value for CMDDIR will be $0005. If you decide to re-format your winchester and make a sub-directory for the system commands, the beginning track/sector address of this sub-directory (e.g. 01/01) must be used. When these four labels have been determined, they can be entered into HIER between lines 49 and 50, and typically will look like :

```
DSKCMD EQU $D22E
DSKCMR EQU $D232
OPNFRD EQU $D254
CMDDIR EQU $0005
```

with any different values discovered substituted for those shown. The modified HIER source can now be re-assembled for use in the STARTUP file as usual.

EOF

Winchester Drive System Software

by: Leo Taylor and Samuel I. Green, Ph.D.

Continued from last month

SEEK and RESTORE

I spent quite a few evenings trying to find the best way to handle the seek and restore commands. The Western Digital hard disk controller is far more intelligent than the floppy interface boards I've worked with. I soon discovered that you never use seek command in a single tasking system like FLEX; the controller does an auto- matic seek when you do a read or write. The restore command, which is used often on a floppy, is almost never needed on a winchester. Since the only read errors I've ever experienced with the winchester were caused intentionally, FLEX never called the restore command in WINDRV. This presents one problem, the restore command tells the controller what seek speed to use. The Zeff-Graves program restored the drive everytime a program exited to FLEX. I didn't want to do a lot of unneeded restores just to set the seek speed, so I came up with a 'trick' to force one restore before the first read command. The controller remembers the seek speed until powered off or reset. To prevent the controller from being reset often, I connected the controller reset (pin 39) to 5 volts as shown in the schematic. A similar modification can be made to the Graves board. If you don't make the modification, and your drive starts to seek slowly (and loudly), you can force a restore by entering WINFMT A followed by two returns.

The seek speed number requires some explanation. My drives came without any documentation, thus I didn't know what seek speed to use. To my surprise, the ideal seek speed to use was zero. When the controller sends the seek pulses out as fast as it can go, a Seagate compatible drive will buffer the pulses and signal the controller when the seek is complete. This worked fine with both the SA-604 and ST-225 drives I tested. If it doesn't work with your system, the equate SEEKSP is in increments of .5 milliseconds.

SELECTING INTERLEAVE FACTOR

Those who have read my interleave article (68 MJ, Oct 85) know that the interleave factor can be adjusted to maximize the read speed of a disk. It will take about an hour to do the following:

1. Format a volume (the shorter the better) with interleave of 1. 2. Copy a 100 sector text file called T.TXT to the winchester. 3. Time how long it takes to do the command LIST T 9999. 4. Record the results and repeat the process, incrementing the interleave factor, until a large drop in time is observed. 5. The fastest time will be the optimum interleave for the list command. Use that number or perhaps one number higher to allow for a safety margin.

The time to list a 100 sector file should be around 4 seconds for a 2 MHZ 6809, or around 7 seconds at 1 MHZ. The interleave number you obtained is not a 'true' interleave factor. 100 sectors in 4 seconds is 40 milliseconds per sector. The disk rotates in 16 milliseconds, so it actually goes around a couple of revolutions plus your interleave factor.

BOOTING FLEX FROM WINCHESTER

WINSYS includes the capability to load an operating system from the hard disk. This bootstrap operation is similar to booting from a FLEX floppy; a small ROM loader loads a larger disk resident loader from track zero sector one. This routine then loads the FLEX.SYS file and jumps to the file's transfer address. This requires several steps to achieve:

1. Add the short (64 byte) bootstrap program WINBOOT to your EPROM monitor. Set the user defined equates to match your system (use the same values as WINTABLE.LIB). The bootstrap on the disk can be loaded anywhere, set BOOTMEM to a location that will not conflict with your FLEX ($C500 is usually OK). 2. Set the equate for DEFALT in WINTABLE.LIB to zero and assemble WINDRV. This will allow FLEX to find STARTUP on drive zero. 3. Append WINDRV to the end of FLEX to make a new FLEX.SYS. This file must be squeezed to eliminate the extra transfer address. 4. Copy your new FLEX.SYS onto your first volume (usually A) and link it. The LINK command will write the starting disk address of FLEX.SYS into bytes 5 and 6 of the bootstrap. 5. Try your the new command in your monitor. The EPROM program should load the bootstrap into BOOTMEM, and that program should load FLEX. Total time is under a second if the disks are up to speed. My drives take 18 seconds from a dead start.

THE LITTLE THINGS THAT COUNT

1. Winchesters under FLEX do not read much faster than properly interleaved floppies, but they write many times faster. This is due to faster rotation speed and lack of verify after write. 2. Though not intentional, WINSYS will operate with two drives of different sizes. Lets say you start out with a 5 meg drive, then add a 15 meg unit. Make the larger unit the lower number drive, and set TRACKS to match the larger drive. The volume sizes should be chosen to fill the larger drive then continue on the smaller drive. Remember that the software will not be able to check for the last volume exceeding the size of the smaller drive.

3. The parking area is actually disk space that the maker of the drive does not certify as usable. I've found I can

'cheat' and format my 160 track drives for 180 tracks and park the heads at track 182. 4. When the controller is reset it selects drive number zero. Unfortunately, this lights up the LED on that drive until WINSYS turns the light out by selecting an unused drive. This can be avoided by jumpering your first drive to select as physical drive 1. 5. After having a silent computer for 10 years, I can't stand the noise from the winchester motors and fan. The enclosed interface schematic includes an extra IC to implement a software power on/off line. This can be used to drive a solid state relay to control the winchester. WINPARK will turn off the motors by writing $80 to WCYLH. A short program can write $00 to WCYLH to power up. This is the second address of the port, and the upper bits are normally unused. 6. The controller must be told by WINDRV to turn off the LEDs after every operation. A side benefit of this is you will be able to tell what the drive is doing by the flashing LED. A seek is brightest, a read is dimmer, a write flickers. 7. The software expects the controller commands to be positive true, not inverted as in the Graves software. Swap the buss transceiver between 74LS640 and 74LS245 if you are currently using inverted data. 8. As mentioned by Graves, the winchesters appear to have a low error rate. WINDRV checks and reports errors, but does not verify after writes. WINFMT does not have any logic to take bad sectors out of the free chain. I've never encountered an error, but if you do there are two alternatives. The TSC utility FLAW can be used to remove a bad sector from the free chain. Or, you can remove bad sections of the disk by creating a dummy volume over the bad spot. Remember, if a sealed drive develops a bad track, you can't replace the disk! 9. WINDRV compares the track number in a read or write command to the size in tracks of that volume, returning an error if too high a track is requested. This prevents a runaway program from clobbering a track on a volume other than the one assigned. 10. Since most systems (including OS-9) insist on formatting a drive starting at track zero, WINSYS provides a means of using a drive for two operating systems. Note the first volume in EXAMPLE B3, which is 80 tracks I've reserved for OS-9. 11. A CRC disk can't be read with ECC enabled, and vice versa. 12. I have not found an 8 head drive to test, but it SHOULD work. There may be a problem with some software accessing sector 0.

WRAP UP

I hope you enjoy using these programs. Due to the small number of FLEX users who can 'BETA TEST' the software for me, there is always a chance for a bug or two to pop up. Let me know if you have any problems, and keep in touch to get any updates I make as time goes by.

I'd like to thank Robert Zeff, David Graves and Phil Gunsel for the early winchester programs that inspired WINSYS.

Leo Taylor

EXAMPLES OF TABLES AND REPORTS

>>> EXAMPLE A1 <<<

```
DEFALT EQU 3 DRIVE FOR FIRST VOLUME
DELAY EQU 10000 TIME OUT DELAY
DRV1ST EQU 0 FIRST PHYSICAL DRIVE
DRVORG EQU $E800 DRIVER GOES HERE
ECFLAG EQU $00 0=CRC $80=ECC (WD1002)
HBASE EQU $E010 FIRST ADDRESS OF PORT
HEADS EQU 4 DEFAULT HEADS/DRIVE
INTERL EQU 16 DEFAULT INTERLEAVE
MAXDRV EQU 3 MAX LOGICAL DRIVE NUM
NODRIV EQU 3 SELECTED FOR LIGHTS OUT
OFFSET EQU $00 USED FOR SPLIT PORT
PRKTRK EQU 181 PARK HEADS HERE
PRECMP EQU 32 PRECOMPENSATION
SEEKSP EQU 0 SPEED NORMALLY=0
TRACKS EQU 160 TRACKS PER DRIVE

SIZA EQU 30
SIZB EQU 70
SIZC EQU 57
SIZD EQU 00
SIZE EQU 00 (REMAINING EQUATES ARE 00)
```

>>> EXAMPLE A2 <<<

```
VOL  DRV  TRK  SIZ     VOL  DRV  TRK  SIZ
 A    0    0   30       B    0   31   70
 C    0  102   57       D    -    -    0

Volume letter to format : A
Number of surfaces (1-8): 4
Interleave factor (1-31): 16
Original name and number: VOLUME-A.EXA 123
Volume Name (CR aborts) : VOLUME-A.NEW
Volume Number (0-65535) : 321

Track being formatted
30

Track being linked
30
```

>>> EXAMPLE A3 <<<

```
VOL     NAME    EXT  SIZ     VOL     NAME    EXT  SIZ
 A    VOLUME-A.NEW   30       B    VOLUME-B.      70
 C    VOLUME-C.TST   57       D    ———————   0
  0--  1--  2--  3-A
Enter drive number and volume letter: 1B
```

>>> EXAMPLE B1 <<<

```
DEFALT EQU 0 DRIVE FOR FIRST VOLUME
DELAY EQU 6500 TIME OUT DELAY
DRV1ST EQU 1 FIRST PHYSICAL DRIVE
DRVORG EQU $CA95 DRIVER GOES HERE
ECFLAG EQU $80 0=CRC $80=ECC (WD1002)
HBASE EQU $E010 FIRST ADDRESS OF PORT
HEADS EQU 4 DEFAULT HEADS/DRIVE
INTERL EQU 1 DEFAULT INTERLEAVE
MAXDRV EQU 9 MAX LOGICAL DRIVE NUM
NODRIV EQU 3 SELECTED FOR LIGHTS OUT
OFFSET EQU $1C USED FOR SPLIT PORT
PRKTRK EQU 181 PARK HEADS HERE
PRECMP EQU 32 PRECOMPENSATION
SEEKSP EQU 0 SPEED NORMALLY=0
TRACKS EQU 180 TRACKS PER DRIVE

SIZA EQU 79 OS-9    .SYS 1
SIZB EQU 02 BASIC   .BAS 1
```

```
SIZC EQU 20 C-SOURCE.C   1
SIZD EQU 08 DICTIONA.RY  1
SIZE EQU 04 EDITOR  .STY 1
SIZF EQU 05 FLEXSORC.ASM 1
SIZG EQU 20 SOURCE  .ASM 1
SIZH EQU 04 WINCHSTR.ASM 1
SIZI EQU 04 FLEX-9  .SYS 1
SIZJ EQU 04 PICTURE .PIC 1
SIZK EQU 01 LIBRARY .LIB 1
SIZL EQU 08 LEOBUG  .ASM 1
SIZM EQU 08 MORESPAC.XXX 1
SIZN EQU 67 WORKDISK.ASM 2
SIZO EQU 10 OS-9WORK.ASM 2
SIZP EQU 04 PICTURE .PIC 2
SIZQ EQU 02 BASIC   .BAS 2
SIZR EQU 20 C-SOURCE.C   2
SIZS EQU 20 SOURCE  .ASM 2
SIZT EQU 08 LEOBUG  .ASM 2
SIZU EQU 04 EDITOR  .STY 2
SIZV EQU 04 FLEX-9  .SYS 2
SIZW EQU 04 WINCHSTR.ASM 2
SIZX EQU 26 MORESPAC.XXX 2
SIZY EQU 00
SIZZ EQU 00
```

>>> EXAMPLE B2 <<<

VOL	DRV	TRK	SIZ	VOL	DRV	TRK	SIZ
A	1	0	79	B	1	80	2
C	1	83	20	D	1	104	8
E	1	113	4	F	1	118	5
G	1	124	20	H	1	145	4
I	1	150	4	J	1	155	4
K	1	160	1	L	1	162	8
M	1	171	8	N	2	0	67
O	2	68	10	P	2	79	4
Q	2	84	2	R	2	87	20
S	2	108	20	T	2	129	8
U	2	138	4	V	2	143	4
W	2	148	4	X	2	153	26

>>> EXAMPLE B3 <<<

VOL	NAME	EXT	SIZ	VOL	NAME	EXT	SIZ
A	OS-9	.SYS	79	B	BASIC	.BAS	2
C	C-SOURCE	.C	20	D	DICTIONA	.RY	8
E	EDITOR	.STY	4	F	FLEXSORC	.ASM	5
G	SOURCE	.ASM	20	H	WINCHSTR	.ASM	4
I	FLEX-9	.SYS	4	J	PICTURE	.PIC	4
K	LIBRARY	.LIB	1	L	LEOBUG	.ASM	8
M	MORESPAC	.XXX	8	N	WORKDISK	.ASM	67
O	OS-9WORK	.ASM	10	P	PICTURE	.PIC	4
Q	BASIC	.BAS	2	R	C-SOURCE	.C	20
S	SOURCE	.ASM	20	T	LEOBUG	.ASM	8
U	EDITOR	.STY	4	V	FLEX-9	.SYS	4
W	WINCHSTR	.ASM	4	X	MORESPAC	.SPC	26

```
  0=-  1=-  2=-  3=-  4=C  5=G  6=L  7=H  8=W  9=-
Enter drive number and volume letter: 4E

+++
```

NAM SQUEEZE.CMD

```
 OPT PAG
 PAG
* SQUEEZE FILE COMPRESSION COMMAND.
*
* SQUEEZE IS USED TO REMOVE EXTRA SPACE
* FROM A FILE. FILES THAT WILL BENIFIT
* FROM SQUEEZING ARE THOSE THAT ARE MADE
* WITH APPEND.CMD OR THOSE WHERE DISKEDIT
* WAS USED TO NULL OUT PART OF A PROGRAM.
* THIS PROGRAM WILL BE OF BENIFIT IF YOU
* HAVE A LIMITED AMOUNT OF SPACE ON YOUR
* SYSTEM DISK AND WANT TO SQUEEZE EVERY
* LAST SECTOR OUT OF IT.
*
* IF TWO OR MORE FILES HAVE BEEN APPENDED
* THERE IS A GOOD CHANCE THE NEWLY MADE
* FILE IS A SECTOR LONGER THAN NEED BE. IF
* THE APPENDED PORTION OVERLAPS PART OF THE
* ORIGINAL (EG ADDING NEW DRIVERS TO FLEX)
* THAN THE ORIGINAL SECTION CAN BE NULLED
* OUT AND THE SPACE RECOVERED WITH SQUEEZE.
* ANOTHER EXAMPLE IS NULLING OUT THE TEXT
* HEADERS AT THE START OF SOME PROGRAMS AND
* USING SQUEEZE TO RECOVER THE SPACE.
*
*
* CALLING FORMAT:
*
*  SQUEEZE   FILENAME.EXT     WORK DRIVE
*  SQUEEZE 2.FILENAME.EXT     DRIVE 2
*
* THE ORIGINAL FILE WILL BE RENAMED TO
* .BAK AND THE SQUEEZED FILE WILL HAVE
* THE ORIGINAL NAME AND EXTENSION. NOTE:
* THE EXTENSION IS REQUIRED TO PREVENT
* ACCIDENTAL SQUEEZING!
*
 PAG
* FLEX EQUATES
*
* ONLY ONE EQUATE NEED BE CHANGED FOR 6809.
*
FLEX EQU $A000 USE $CD00 FOR 6809

BUFPNT EQU FLEX+$C14
WARMS EQU FLEX+$D03
PSTRNG EQU FLEX+$D1E
GETFIL EQU FLEX+$D2D
RPTERR EQU FLEX+$D3F
FMS EQU FLEX+$1406

 ORG FLEX+$100
 SPC 3
* START OF PROGRAM
*
START BRA START2
VN FCB 2 VERSION NUMBER
COUNT FCB 0 BYTE COUNTER
TEMP FDB 0
FLAG FCB 0 FOUND STARTING ADDR
 SPC 1
START2 LDX BUFPNT
 STX TEMP TO BE REUSED
 LDX #FCB2 POINT TO DESTINATION FCB
 JSR GETFIL
 BCS ERROR
 TST 12,X ANY EXTENSION?
 BNE EXTOK
 JMP EXTERR

EXTOK LDX TEMP
 STX BUFPNT RESTORE POINTER
 LDX #FCB1 POINT TO SOURCE FCB
 JSR GETFIL REREAD FILE NAME
 LDX TEMP
 STX BUFPNT
 LDX #FCB1+49 SCRATCH BYTES AREA
 JSR GETFILE
```

```
 LDAA #'B BAK EXTENSION
 STAA 12,X
 LDAA #'A
 STAA 13,X
 LDAA #'K
 STAA 14,X
 LDX #FCB1
 LDAA #13 RENAME FUNCTION
 STAA 0,X
 JSR FMS
 BNE ERROR
 LDX #FCB1 SOURCE FILE
 LDAA #1
 STAA 0,X
 JSR FMS
 BNE ERROR
 CLR 0,X SET FOR READ
 LDX #FCB2
 LDAA #2
 STAA 0,X
 JSR FMS OPEN FOR WRITE
 BNE ERROR
 CLR 0,X SET FOR WRITE
 CLR FLAG DEFAULT TO TEXT
GETYPE LDX #FCB1
 JSR FMS
 BNE ERROR
 TSTA NULL?
 BEQ GETYPE
 CMPA #2 BINARY FILE ?
 BEQ BINARY YES,BINARY FILE TYPE
 CMPA #$16 STARTING ADDRESS?
 BEQ STRADR PROCESS STARTING ADDRESS
 JMP TEXT NOT 0 OR 2 OR 16
 SPC 3
* MULTI-PURPOSE ERROR SECTION
*
ERROR LDAA 1,X GET ERROR NUMBER
 CMPA #8 IS IT 8?
 BNE ERR8AD ERROR 8 IS EOF
 TST FLAG
 BEQ ERROK
 LDAA #$16 STARTING ADDRESS MARK
 LDAB #1
 STAB COUNT
 BSR WRITE
 LDAA TEMP
 INC COUNT
 BSR WRITE
 LDAA TEMP+1
 INC COUNT
 BSR WRITE
 BRA ERROK

ERR8AD JSR RPTERR
ERROK LDAA #$04 CLOSE FUNCTION
 LDX #FCB1
 STAA 0,X
 JSR FMS
 LDAA #$04 CLOSE FUNCTION
 LDX #FCB2
 STAA 0,X
 JSR FMS
WARMS2 JMP WARMS

EXTERR LDX #NOEXTM
 JSR PSTRNG
 BRA WARMS2
 SPC 3
* BINARY FILE SECTION
*
BINARY LDAB #$FF
 STAB FLAG INDICATE BINARY
 STAB FCB1+59 TURN OFF COMPRESSION
 STAB FCB2+59
 LDAB #1
 STAB COUNT
 BSR WRITE OUTPUT 2
 LDAB #3 OUTPUT AOR AND COUNT
 STAB COUNT BYTE COUNT
 BSR REDWRT
 STAA COUNT BYTE COUNT
 BSR REDWRT OUTPUT DATA BLOCK
GETYP2 BRA GETYPE
 SPC 3
* START ADDRESS SECTION
*
STRADR JSR FMS GET START ADR
 BNE ERROR
 STAA TEMP SAVE ADR
 JSR FMS
 BNE ERROR
 STAA TEMP+1
 BRA GETYP2
 SPC 3
* READ AND WRITE SECTION
* READS THEN WRITES BYTE 'COUNT' TIMES
*
REDWRT LDX #FCB1 READ FILE
 JSR FMS GET BYTE
 BNE ERROR2
 SPC 1
WRITE LDX #FCB2 WRITE FILE
 JSR FMS WRITE BINARY BYTE
 BNE ERROR2
 DEC COUNT
 BNE REDWRT
 RTS
 SPC 1
ERROR2 INS FIX STACK
 INS
 JMP ERROR
  SPC 3
* TEXT FILE PROCESSING
*
TEXT BSR WRITE WRITE FIRST CHARACTER
TLOOP LDAA #255 MAXIMUM COUNT
 STAA COUNT
 BSR REDWRT
 BRA TLOOP LOOP TILL REDWRT EXITS
 SPC 3
* MESSAGES
*
NOEXTM FCC 'EXTENSION REQUIRED'
 FCB 7,4

FCB1 RMB 320
FCB2 RMB 320

 END START

+++
```

Bit-Bucket

By: All of us

"Contribute Nothing · Expect Nothing", DMW '86

MICRONICS RESEARCH CORP.

Micro-computers - Hardware and Software
GIMIX® Sales, Service and Support

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA. V2S 1E2

Dear Don,

Here's a little trick that I'm sure every BASIC programmer will find **very** useful. It involves the PTR function of XBASIC, normally one of the most useless of all of XBASIC's functions.

Here's how it's supposed to work. Let's say you have a program with lots of references to a variable named A%. By typing in 'PRINT PTR(A%)' in response to the READY prompt, XBASIC will return an address (in decimal), which points to where the current value of A% is stored on the Variable-Stack. Now suppose you wished to change the value (from 5, let's say, to 7), you could enter 'DPOKE xxxx,7', where xxxx is the decimal address just identified. This is about the only use for the PTR function that I can see, but as it's so much easier to simply enter 'A%=7', why bother with PTR at all? On the other hand, if the variable were Floating-Point (let's say A), the **only way to change its value** is via the primitive 'A=whatever'. Similarly with String-variables!

However, on the Stack where all these variables are stored, 2 bytes below the variable's value is the variable-name. So what? you ask. The important thing is that, even though you may have 1000 references to A% scattered throughout your program, **this is the ONLY place where its name is recorded!** The individual program-lines simply refer to a particular Stack-location to identify the name of a specific variable.

Therefore, if, in the interests of meaningful variable-names, let's assume you wish to change A% to B%, it's only necessary to enter

```
POKE PTR(A%)-2,66
```

(where 66 is the decimal equivalent of the HEX-ASCII 42, ie, B), and hey presto! - - ALL references to A% in your program have now been changed to B%!! Further, unlike using an editor, it **does not** change AA%, BA%, etc., into AB%, BB%.

Be careful if you wish to change A% into BC%. You would now have to

```
DPOKE PTR(A%)-2,66*256 + 67
```

to ensure that 'B' gets stored in the most-significant byte, and 'C' in the least significant. Note that we use DPOKE (not POKE) to store the 2 characters!

Similarly if you wish to go from a 2-character name to a single character, let's say to reverse the above process and change BC% to A%, we'd enter

```
DPOKE PTR(BC%)-2,65*256
```

to ensure, not only that the single-character 'A' occupies the MS-byte, but that a NUL (ie, 00) gets stuffed in the LS-byte to cancel the former occurrence of 'C'.

Of course, we're not restricted to Integer variables alone. We can apply exactly the same process to change the name of a Floating-point variable, or even a String. Due to the structure of the Stack, however, we cannot change the variable-**type**. That is, we can't change M$ to M%, or X% to X. Neither will this procedure work for subscripted variables - in fact, I wouldn't recommend even trying it, unless you're prepared to risk blowing up your program!

The major differences between RBASIC (commencing with Version 2.4) and XBASIC are that RBASIC's PTR function points **directly** to the variable-name (so it's not necessary to subtract 2 from the returned address), and also works for **all** variables, whether subscripted or not. Naturally, if you wish to point to the variable-**value** of unsubscripted integer or floating-point variables you'll have to add 2 to this address, but why anyone would wish to do this I just can't imagine!! Has anyone out there **ever** used PTR for any useful purpose, other than idle curiosity as to how BASIC stores its variable-values?

Anyway, you now have a GLOBAL VARIABLE-NAME CHANGER for unsubscripted variables, which you can use directly from within BASIC itself. Hope you like it!

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Sincerely,

Bob

R. Jones
President

PS Have you seen Peripheral Tech's flyer, which mentions comparison-tests between the 68000 version of RBASIC on their PT68K-2 and Microsoft's GWBASIC running on an IBM AT? RBASIC is **five to ten times as fast!!!** Wow! I can't believe it myself, especially as I haven't even begun to optimise it for speed yet!

APS IT GOES WITHOUT SAYING THAT YOU SHOULD BE VERY CAREFUL NOT TO CHANGE TO A VARIABLE-NAME THAT'S ALREADY IN USE!

6,Spencer Walk
Rickmansworth
Hertfordshire
WD3 4EE
United Kingdom

Dear Don,

Here is another little contribution for the BIT BUCKET. Some long way back somebody gave me a utility named DPRINT. Basically this takes a line of text from the keyboard and outputs it directly to the printer. Nothing much really, but when I got my PT68K-2, there was nothing like this distributed with the standard software. So, I hacked together a look-alike.

I find this useful when I want to address an envelope, or make a label. It just turns the printer into the equivalent of a typewriter, simple but effective, as all the best ideas are. If the guy who wrote the original code is still a reader I hope he is not offended by my efforts.

Best regards

John Pink

```
LISTING OF FILE        QP        TIME 21:56:04   DATE 07/05/88   PAGE 0    1

00001 00000000                   ****************************************
00002 00000000                   * "QP"    WILL DIRECTLY PRINT A LINE
00003 00000000                   *             OF TEXT TO THE PRINTER.
00004 00000000                   *
00005 00000000                   * JOHN PINK - JUNE 1988 - Based on a 6809 FLEX utility,
00006 00000000                   *                        source unknown.
00007 00000000                   *
00008 00000000                   * Modified to avoid use of GETNIT which suppresses spaces
00009 00000000                   * JULY 1988  - JOHN PINK
00010 00000000                   *
00011 00000000                   * SYNTAX: QP  The are no arguments
00012 00000000                   ****************************************
00013 00000000                   *
00014 00000000                   * SK*DOS EQUATES
00015 00000000                   *
00016 00000000 0000A032          OCNTRL  EQU     $A032
00017 00000000 0000A02C          INLINE  EQU     $A02C
00018 00000000 0000A033          PUTCH   EQU     $A033
00019 00000000 0000A034          PCRLF   EQU     $A034
00020 00000000 0000A035          PSTRNG  EQU     $A035
00021 00000000 0000A000          VPOINT  EQU     $A000
00022 00000000 0000A01E          WARMST  EQU     $A01E
00023 00000000                   *
00024 00000000                   * DATA AREA EQUATES
00025 00000000                   *
00026 00000000 00000268          LINBUF  EQU     $68
00027 00000000 00000CFE          PAUSED  EQU     3326
00028 00000000                   *
00030 00000000                   *
00031 00000000                   *PROGRAM START
00032 00000000                   *
00033 00000000 00000000                  ORG     $0000   POSITION INDEPENDANT    I HOPE!
00034 00000000 60000002          START   BRA     BEGIN
00035 00000004 00000004          VERSION         0100 .
00036 00000004                   *
00037 00000004 A000              BEGIN   DC      VPOINT  POINT TO USRFCB .
00038 00000006 43EE0CFE                  LEA     PAUSED(A6),A1 SET CURRENT STATUS OF OUTPUT
 PAUSE
00039 0000000A 13D100000148              MOVE.B  (A1),PSAVE SAVE IT LOCALLY
00040 00000010 422E0CFE                  CLR.B   PAUSED(A6) CLEAR OUTPUT PAUSE
00041 00000014 A034                      DC      PCRLF   SPACE THINGS OUT
00042 00000016 49FA0058                  LEA     HEADER(PC),A4 LOAD RULER
00043 0000001A A035                      DC      PSTRNG  SEND IT TO TERMINAL
00044 0000001C                   *
00045 0000001C 183C002A          PROMPT  MOVE.B  $2A,D4 LOAD A '*'
00046 00000020 A033                      DC      PUTCH
00047 00000022 A033                      DC      PUTCH
00048 00000024 A033                      DC      PUTCH   SEND IT TO FORM A NEW PROMPT ***
00049 00000026 A02C                      DC      INLINE
00050 00000028 6114                      BSR.S   GET     OK NOW CHECK IT AND PRINT IF NOT C
R.
00051 0000002A                   *
00052 0000002A 1219              MORE    MOVE.B  (A1)+,D1 GET SOME MORE TO PRINT
00053 0000002C B23900000000D             CMP.B   $0D,D1  HAS THE END OF LINE BEEN REACHED?
00054 00000032 66E8                      BNE.S   PROMPT  YES?,PUT UP A NEW PROMPT
00055 00000034                   *
00056 00000034 43EE0CFE          EXIT    LEA     PAUSED(A6),A1 ALL DONE, GET ADDRESS OF PAU
SE
00057 00000038 12BA0113                  MOVE.B  PSAVE(PC),(A1) RESTORE THE OLD PAUSE
00058 0000003C A01E                      DC      WARMST  AND EXIT TO SK*DOS
00059 0000003E                   *

LISTING OF FILE        QP        TIME 21:59:56   DATE 07/05/88   PAGE 0    2

00060 0000003E 43EE0260          GET     LEA     LINBUF(A6),A1
00061 00000042 1219                      MOVE.B  (A1)+,D1     MOVE THE NEXT BYTE
00062 00000044 0C01000D                  CMP.B   $0D,D1 IS IT A CR?
00063 00000048 6700FFEA                  BEQ     EXIT    IF SO ALL IS DONE, SO EXIT
00064 0000004C                   *
00065 0000004C 383CFFF2          SEND    MOVE.W  $FFF2,D4 LOAD NEW OUTPUT DEVICE #
00066 00000050 A032                      DC      OCNTRL  OUTPUT SWITCH, TO THE PRINTER
00067 00000052                   *
00068 00000052 1801              PRINT   MOVE.B  D1,D4   MOVE THE CUARENT CHARACTER AND
00069 00000054 A033                      DC      PUTCH   OUTPUT IT TO PRINTER
00070 00000056 1219                      MOVE.B  (A1)+,D1     AND GET THE NEXT CHARACTER
00071 0000005B 0C01000D                  CMP.B   $0D,D1 IS IT A CR?
00072 0000005C 66F4                      BNE.S   PRINT   NO?, SO PRINT IT
00073 0000005E 1801                      MOVE.B  D1,D4   YES? SO PRINT CALF
00074 00000060 A033                      DC      PUTCH
00075 00000062 183C000A                  MOVE.B  $0A,D4 AND SEND A LINE FEED TO PRINTER
00076 00000066 A033                      DC      PUTCH
00077 00000068                   *
00078 00000068 383CFFF0          RESET   MOVE.W  $FFF0,D4 LOAD TERMINAL DEVICE #
00079 0000006C A032                      DC      OCNTRL  OUTPUT SWITCH, BACK TO TERMINAL
00080 0000006E 4E75              RETURN  RTS     AND     RETURN
00081 00000070                   *
00082 00000070                   * MESSAGES AND HEADERS
00083 00000070                   *
00084 00000070 515349434B203052494E HEADER  DC.B    "QUICK PRINT",$0D,$0A
               540D0A
00085 0000007D 455A455259204C494E45          DC.B    "EVERY LINE IS SENT TO THE PRINTER AFTER'
$0D,$0A
               2049532053454E542054
               4F20544852205052494E
               5445522041465445520D
               0A
00086 000000A6 275245544552524E272049        DC.B    "'RETURN' IS PRESSED. EXIT TO SK*DOS BY PR
ESSING",$00,$0A
               5320503245553534442E
               204558495420544F2053
               4B2A444F532042592050
               524553534945470D0A
00087 000000D7 275245544552524E272041        DC.B    "'RETURN' AT THE START OF THE LINE",$0D,$0
A
               5420544B452053544152
               54204F4620544B45204C
               494E450D0A
00088 000000FA 202D2D2020202D2D2D3A          DC.B    "---------:---------:---------:---------:-
---------:---------:---------:------
               2D2D2D2020202D2D2D3A                    ---:",$0D,$0A,$04
               202D2D2D2D2D2D2D2D3A
               20202D2D2D20202D2D3A
               2D2D2D2D2020202D2D3A
               2D2D2D2D2D2D2D2D3A
               2D2D2D2D2D2D2D2D3A
               2D2D2D2D2D2D2D2D3A
               0D0A04
00089 0000014D                   *
00090 0000014D                   *
00091 0000014D                   * SAFE KEEPING
00092 0000014D 00000001          PSAVE   DS.B    1
00093 0000014E                   *
00094 0000014E 00000004                  END     BEGIN

000 Syntax Error(s)
```

PRESS RELEASE

THE SOFT CENTRE OPEN U.S. OFFICES

As a result of increased sales in the U.K. and Europe of some of the most exciting products to hit the OS-9 world, the Soft Centre announces the opening of Windsor Systems. Windsor Systems is owned by Steven Weller, Steve is a senior software engineer and has been employed by the Soft Centre for the last four years.

Software products now available in North America with full support include:-

GKS - Graphical Kernel System to level 2c.

DISK CACHING - make your computer run three times faster, this is the typical performance improvement on a 68020 Computer with hard disk.

SCSI - Device Driver System for using SCSI with OS-9, offering separate logical and physical drivers, the SCSI driver system creates a flexible interface for all your SCSI drivers. Supports SCSI commands disconnect and reconnect.

TAPE ARCHIVING - Integrated bulk storage archiving utility, incorporating UNIX Tar and Cpio compatibility.

VBF - Variable Block File Manager, a communications orientated file manager which brings out the best in multichannel intelligent interface cards.

MCF - Multi-Characted File Manager is the same as VBF, but additionally allows the use of line editing functions on read line and write line.

VIVANET - Vivanet is a serial port driver for the Network File Manager.

MATH - Math trap handler and math library for the 68008, 68000 and 68010 using the MC68881 as a peripheral.

For more information contact:-

Windsor Systems
2407 Lime Kiln Lane
Louisville
Kentucky 40222
U.S.A.

Tel: 502-425-9560
Fax: 502-425-6853

The Soft Centre
Software House
Burr Street
LUTON Bedfordshire
LU2 OHN England

Tel: 011-44-582-405511
Fax: 011-44-582-456521

WINDRUSH
Micro Systems Ltd

PLµS-68K ... A Complete Programming Environment for OS-9

Windrush Micro Systems Limited are pleased to announce the immediate availability of PLµS-68K (Programming Language for µ (micro) Systems), a totally Inlegrated programming environment for the Motorola MC68000 family of Microprocessors.

PLµS-68K is a complete programming environment for OS-9/68K system users. The product comprises a co-resident EDITOR - COMPILER -and source tevel DEBUGGER which provide the ultimate in productive working environments. You can Edit a program, immediately call the Compiler and then immediately call the Debugger ... without saving files to disk or returning to the operating system.

PLµS is modelled on Pascal with many of the useful features of 'C' included. Variable types range from signed and unsigned bytes through to 32-bit fixed and floating point numbers. The '020/881' version extends the range to 84-bit floating point numbers. Low level programming facilities, such as direct access to registers, simplifies the interface to assembly language programs and operating system calls. The single pass compiler compiles code at 20,000 lines of source per minute with output code efficiency second only to assembly language.

PLµS was designed from the ground up to produce code for stand-alone 68xxx targets and as such does not carry any license requirements for the object code produced. The system is equally adept at producing modules for an OS-9 environment. *A unique feature of the language is the ability to produce OS-9 device drivers and descriptors* ... the last domain of the assembly language programmer!

The product is available in two versions. PLµS-68K which produces code for the MC68000, 008, 010 and 020 processors and PLµS-020 which includes PLµS-68K and also produces code for the MC68020 with an MC68881 math co-processor.

For further information contact Bill Dickinson at (0692) 404086

MOTOROLA INC.

Microprocessor Products Group
8501 William Cannon Drive West
Austin, Texas 78735-8598

CONTACTS
Shemaz Daver
Cunningham Communication, Inc.
(408) 982-0400

Dean Mosley
Microprocessor Products Group
(512) 440-2839

MOTOROLA ANNOUNCES 33 MHZ 68882 MATH COPROCESSOR

Apollo is First to Announce Plans to Incorporate New 882 in Workstations

AUSTIN, Texas, July 11, 1988 -- Motorola today announced that the company's 68882 (882), its second-generation 32-bit floating-point math coprocessor, is now available at a speed of 33 MHz. The 33 MHz chip enhances system performance by increasing the processing speed of mathematical operations that are crucial to business and engineering environments. Apollo Computer Inc. (Chelmsford, Mass.) today became the first company to announce a 68030-based workstation incorporating the 33 MHz coprocessor.

Floating-point coprocessors are used to speed mathematical calculations in a wide range of applications. A coprocessor can perform mathematical functions 300 times as fast as software solutions. The 882 coprocessor is a high-performance single chip used with Motorola's 68000 family that includes the 68000, 68010, 68020 and 68030 microprocessors. The 882 conforms to the IEEE Standard for Binary Floating Point Arithmetic and offers software and pin compatibility with its predecessor, the 68881 (881).

"Floating-point functionality is becoming a standard feature in today's computers," said Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "The 882's hardware and software compatibility with the 68000 family ensures the performance and upgrades that customers expect of Motorola."

The 33 MHz 882 is the first single chip to break the two million Whetstone barrier. (The Whetstone is a standard benchmark that tests a processor's ability to perform mathematical operations.) The 68881, the 882's predecessor, was the first single chip floating-point coprocessor to break the one million Whetstone barrier. System users can simply unplug the 881 and replace it with an 882 chip, gaining a 50 percent performance increase. With optimized software and the new 33 MHz clock speed, the 882 can provide a two to four-fold performance increase over the 881. The 33 MHz 882 is priced at $708 in single quantities and is available 60 days at receipt of order.

MOTOROLA INC.

CONTACTS
Shemaz Daver
Cunningham Communication, Inc.
(408) 982-0400

Dean Mosley
Microprocessor Products Group
(512) 440-2839

APOLLO ANNOUNCES HIGH-END WORKSTATIONS BASED ON MOTOROLA'S 68030

68000 Continues to Provide Best Price-Performance Ratio

BOSTON, July 11, 1988 — Motorola and Apollo Computer Inc. today jointly announced the incorporation of Motorola's 68030 (030) microprocessor in two new Apollo workstations. The Series 3500 Personal Workstation and Series 4500 Personal Super Workstation both utilize the 030 chip as the central processor. Also, both workstations use Motorola's 68882 (882) floating-point math coprocessor.

The Series 3500 Personal Workstation is the first color workstation operating at four MIPS (millions of instructions per second) to sell for less than $10,000. The $8,000 workstation uses the 25 MHz 030 as a central processor and a 25 MHz 882 coprocessor. The Series 4500 performs at seven MIPS and sells for $19,000. It incorporates the 33 Mhz 030 and a 33 MHz 882 coprocessor.

Motorola's 030 provides a fully integrated architecture that drives down the price-performance ratio of computing systems. The chip is the first to implement a dual bus architecture, CPU engine, memory management, and instruction cache and data cache on a single chip. In addition, the 030 maintains complete compatibility with the entire Motorola 68000 family line, ensuring easy migration of application software.

"With today's announcement, Apollo is demonstrating that no one can touch the price-performance ratio provided by the 68000 family," said John Mitchell, vice chairman of Motorola. "There should be no doubt that the 68000 family will continue to dominate the workstation market."

The 030 is the latest addition to Motorola's 68000 family line which includes the 68000, 68010 and 68020. According to Dataquest, a San Jose-based research firm, Motorola has shipped the most 32-bit microprocessors in the world. The 68000 family is used in a wide range of applications such as supercomputers, high-end workstations, business computers and embedded control devices.

In 1981, Apollo introduced the world's first workstation, DN100, which was based on two 68000 microprocessors. In its seven years of developing workstations, Apollo has continued to use the 68000 family in its products. Users of Apollo's new line of workstations protect their original investment in software as the 030 is compatible with the 68000 family line.

Motorola's $2.2 billion Semiconductor Products Group Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola, Inc. The company is the largest and broadest supplier of semiconductors in North America, with a balanced portfolio of more than 50,000 devices.

MOTOROLA INC.

EDITORIAL CONTACT:
Angela Hatfield
(602) 994-6019

READER CONTACT:
Training & Technical Operations
Motorola, Inc. -- HW68
1140 S. Priest Drive
Tempe, AZ 85281
(602) 994-6900

INQUIRY RESPONSE:
Technical Information
P.O. Box 52073
Phoenix, AZ 85072

NEW MC68000 VIDEO SERIES PROVIDES FLEXIBLE, COMPREHENSIVE TRAINING

Phoenix, Arizona, July 18, 1988 . . . Motorola's Semiconductor Products Sector has announced availability of its first video training series. The MC68000 Video Training Series (MTTV2) is an ultra-sophisticated video training system for the MC68000 microprocessor. The series is organized into 18 modules, with one video tape per module to offer the utmost flexibility in training. It also includes five Student Packs with corresponding in-depth workbooks and self-evaluations. An optional Instructor Pack and an optional MC68000 Educational Computer Board are also available.

Based on Motorola's instructor-led training courses, the series uses colorful computer-generated graphics and integrated study materials to make learning easy. The series is a fast, flexible way for a company to train its people while saving thousands of dollars in time and training expenses. The course is designed to fit individual learning requirements, environments and schedules.

Students can virtually train themselves at their own speeds and to meet their particular needs by using the Module Dependency Map in the Course Guide. The modules are also ideal as a group lecture series. Estimated average time to complete the course is 40 hours, which includes approximately 40% text activity and 60% videotape activity.

Motorola's Video Training Series is available now. Pricing for the 18-tape MC68000 Video Training Series complete with five Student Packs and applicative materials is $9,000. The optional Instructor Pack is available for $175. The Educational Computer Board costs $495 with a switching power supply available for $215 and a cable assembly available for $85. All pricing is in U.S. dollars for U.S. delivery only.

To order the MC68000 Video Training Series (MTTV2), or to arrange for a screening of the videotapes at your local Motorola sales office, call Training & Technical Operations toll-free at 1-800-521-6274.

MC68000 VIDEO TRAINING SERIES COURSE DESCRIPTION

Video Tape Series

1. Introduction
2. Programming Model
3. Hardware Overview
4. Basic Addressing Modes
5. Basic Instructions
6. Programming Problems
7. System Control Pins
8. Exception Concepts
9. External Exceptions
10. Internal Exceptions
11. Miscellaneous Instructions
12. Example Programs
13. DMA Control and Synchronous Bus
14. Advanced Addressing Modes
15. Advanced Instructions
16. Advanced Example Programs
17. Review
18. MC68000 Educational Computer Board Lab

Student Packs

Each Student Pack contains a Course Guide, Workbooks 1-18 corresponding to the video modules, a Lab Experiments Workbook, a User's Manual, Programming Cards, and a Reference Manual with technical summaries, application notes and article reprints. Extra student packs may be ordered as needed at an additional cost of $150 each, in U.S. dollars for U.S. delivery only. Quantity discounts are also available.

Optional Instructor Pack

The Instructor Pack contains an Instructor Guide and master copies of workshop diagrams from which to make transparencies, in addition to the materials listed in the Student Pack. Answers are also printed with the questions.

Optional Educational Computer Board

The Optional Educational Computer Board is available with or without power supply for purchase separately.

###

CONTACTS
Shernaz Daver
Cunningham Communication, Inc.
(408) 982-0400

Dean Mosley
Microprocessor Products Group
(512) 440-2839

MOTOROLA ANNOUNCES SOFTWARE CATALOG FOR 88000 AND 68000 MICROPROCESSORS

Catalog Contains Largest 32-bit Software Base

AUSTIN, Texas, July 25, 1988 -- Motorola today announced that a software catalog for its 88000 and 68000 microprocessor families is now available. The catalog, called The Source, lists more than 300 software products, representing the largest installed base of 32-bit software.

The Source is published specifically for software developers, systems designers and end users. It describes a wide range of operating systems, language software, development support software, and software applications for graphics, communications and business. The catalog also includes descriptions of emulation software products that allow Motorola customers to run MS-DOS applications on 88000 and 68000 systems.

"The 68000 family has created a large 32-bit software base that covers the broadest base of software applications," said Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "The increasing software support for the 88000 boosts our lead in the 32-bit software market and gives our customers a significant competitive advantage."

More than 25 companies are developing software products for the 88000, Motorola's RISC microprocessor design. Announced in April 1988, the 88000 is a general-purpose microprocessor line that powers systems in the business, engineering and embedded-control markets.

The 68000 microprocessor family consists of the 68000, 010, 020 and 030. According to Dataquest, a San Jose-based research firm, the 68000 family has the largest installed 32-bit hardware and software base. The compatibility of the microprocessors in the 68000 family allows software to easily migrate from one processor to another.

The Source lists software packages that support Motorola's 68881 and 68882 floating-point math coprocessors and the 68851 paged memory management unit. The catalog is available at no charge from the Motorola Microprocessor Products Group. To obtain the catalog, write Motorola Literature Distribution Center, P.O. Box 20912, Phoenix, AZ 85036, order number BR506 -- The Source.

GENERAL MICRO SYSTEMS
INCORPORATED

For further information:
Scott Bowman (714) 625-5475

FOR IMMEDIATE RELEASE

PHOTOS ENCLOSED
B&W and Color

DUAL 68020-BASED 8-CHANNEL SERIAL I/O PROCESSOR FOR VMEBUS

Montclair, Calif, August 5, 1988 – A VMEbus board with a 32-bit, 25 MHz 68020 CPU for multiprocessing and another 68020 for independent serial I/O processing, is now available from General Micro Systems Inc.

The GMSV07-SIO-1provides eight multiprotocol serial channels, with up to 16 KBytes of buffer memory per channel, controlled by an independent 68020, on a SAM™ (Special Application Module) mezzanine module. This allows data to be written to or read from the board through the P2 connector, without taking up time on the VMEbus. This supports increased system throughput while decreasing VMEbus traffic.

The board also is a 68020-based CPU card with a 68881 co-processor for full 32-bit operation and up to 1MByte of no-wait-state, dual-ported SRAM and two serial channels to support high speed multi-processing.

Up to 512KByte of EPROM (128K of EEPROM) for system and for program codes is also available. The second CPU provides the equivalent of DMA capability on eight serial I/O channels, which support asynchronous operation up to 9600 baud simultaneously on all channels.

The GMSV07-SIO-1 has a 28536 programmable configuration controller with timers and a 68155 bus interrupt manager. Using mailbox/location monitor interrupts for inter-processor signalling, it supports real-time multiprocessing.

The SAM local bus module allows data transfers to/from on-board buffer memory to external devices. This allows data to be handled through the serial channels without taking up CPU time or increasing traffic on the VMEbus. All I/O lines run through the P2 connector, simplifying cabling and board removal from the card cage. "D" connectors to match DCE or DTE configurations are available for RS232, RS422 or RS485 applications.

The intelligent serial SAM module also adds up to 256 KBytes of onboard buffer RAM to that on the main board, significantly speeding up data transfers. This buffer RAM can be segmented into sixteen 16KBytes, with each channel thus supporting data transfers at very high rates without taking CPU time to load/unload the buffers. The SAM module can optionally include up to 256 KBytes of EPROM.

The CPU board can both originate and service interrupts on the VMEbus. Distributed interrupt handling dynamically allocates interrupt handling functions between processors on the bus.

The GMSV07-SIO-1 is a double Eurocard 9.2" x 6.3" (234mm x 160mm) with a mezzanine SAM serial I/O module, requiring a single slot in a VMEbus card cage. For applications not requiring highest speeds, the second 68020 on the SAM module may be omitted. The board also is available for extended temperature applications.

The GMSV07-SIO-1 is OEM priced beginning at $3142, is available from stock, and carries a two-year warranty on parts and labor.

General Micro Systems Inc., located at 4740 Brooks St., Montclair, California 91763, telephone (714)625-5475, FAX 714-621-4400, has been providing reliable, high performance, designed and manufactured in the U.S.A., microcomputer modules since 1978, and offers a full line of modules to VMEbus specifications.

# # #

Classifieds As Submitted - No Guarantees

MUSTANG-020 20Mhz with 68881. OS9 Professional Package & C $3500. Call Tom (615)842-4600.

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

S+System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275. 3-Dual 8" drive enclosure with power supply. New in box. $125 each. 5-Siemens 8" Disk Drive, $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495. **QUME QVT-102** terminal, likenew, amber screen $250. or bestoffer.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09CPU Card- $900 complete.
Tom (615) 842-4600 M-F 9AM to 5PM EST

***

SWTPC 6809 System S709, Dual 8" drives, 8212 terminal, 128 K RAM, Okidata wide carriage, with tractor feed, Make an offer. (813) 462-0511 or (813) 536-0018. Pete Yore.

THE SOFT CENTRE OPEN U.S. OFFICES

As a result of increased sales in the U.K. and Europe of some of the most exciting products to hit the OS-9 world, the Soft Centre announces the opening of Windsor Systems. Windsor Systems is owned by Steven Weller, Steve is a senior software engineer and has been employed by the Soft Centre for the last four years.

Software products now available in North America with full support include:-

GKS ... Graphical Kernel System to level 2c.

DISK CACHING ... make your computer run three times faster, this is the typical performance improvement on a 68020 Computer with hard disk.

SCSI ... Device Driver System for using SCSI with OS-9, offering separate logical and physical drivers,the SCSI driver system creates a flexible interface for all your SCSI drivers. Supports SCSI commands disconnect and reconnect.

TAPE ARCHIVING ... Integrated bulk storage archiving utility, incorporating UNIX Tar and Cpio compatibility.

VBF ... Variable Block File Manager, a communications orientated file manager which brings out the best in multichannel intelligent interface cards.

MCF ... Multi-Character File Manager is the same as VBF, but additionally allows the use of line editing functions on read line and write line.

VIVANET ... Vivanet is a serial port driver for the Network File Manager.

MATH ... Math trap handler and math library for the 68008, 68000 and 68010 using the MC68881 as a peripheral.

For more information contact:-

Windsor Systems
2407 Lime Kiln Lane
Louisville
Kentucky 40222
U.S.A.

Tel: 502-425-9560
Fax: 502-425-6853

The Soft Centre
Software House
Burr Street
LUTON Bedfordshire
LU2 0HN England

Tel: 011-44-582-405511
Fax: 011-44-582-456521

ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage for the **Atari & Amiga™** series of 68000 computers.

Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari & Amiga computers. Especially so since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari & Amiga.* As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard ☐ VISA ☐

Card #_____ Exp. Date_____

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $_____

Name _____

Street _____

City _____ State ____ Zip _____

Country _____

My Computer Is: _____

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

***Foreign Surface: Add $12.00 per Year to USA Price.**

***Foreign Airmail: Add $48.00 per Year to USA Price.**

***Canada & Mexico: Add $9.50 per Year to USA Price.**

***U.S. Currency Cash or Check Drawn on a USA Bank !**

68 Micro Journal
5900 Cassa dra Smith Rd.
POB 849
Hixson, TN 37343

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst.& fixes, Prime, *Pmod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Harkness.
Disk- 8	Crtest, Lampher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven Language Recognition Utility, Anderson March1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986, (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - 68MJ 4/88Korpi.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9 FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:

~~$199.00~~

• 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

• 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

• 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up in the *"Learn a New Language"* syndrome - Write Your Program in BASIC, Debug It in BASIC and Then Compile It to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:

$69.95

SPECIAL

Thank-You-Sale

Only From:

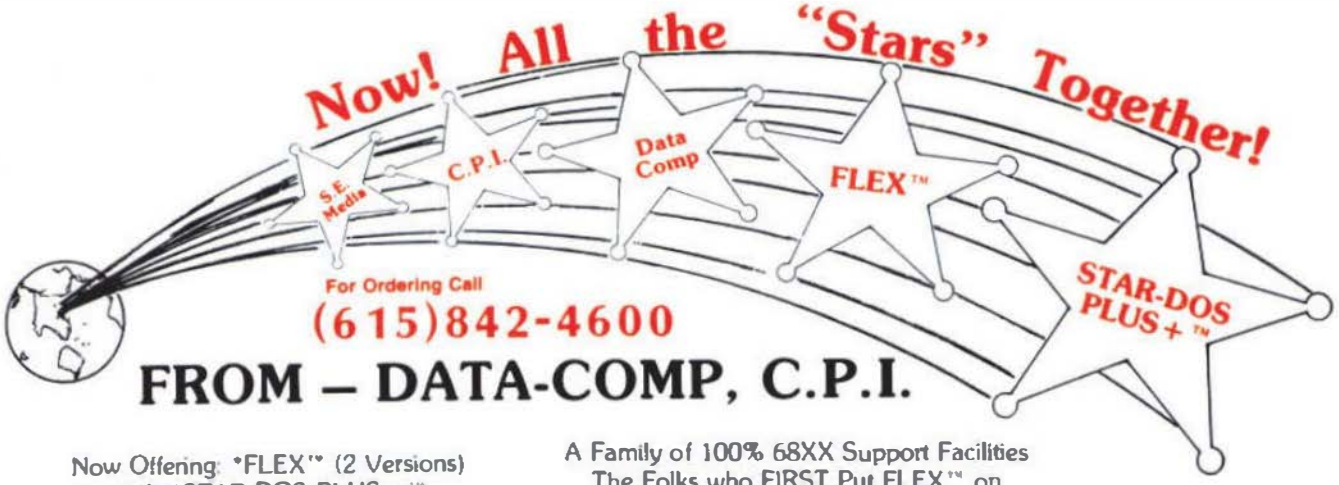
CPI **S.E. Media™**

5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - ~~6800-6809-68000~~

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.

Price subject to change without notice.



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K

FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actually a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed *whiz on disk I/O.* Fact is, the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks!

SALE!!!

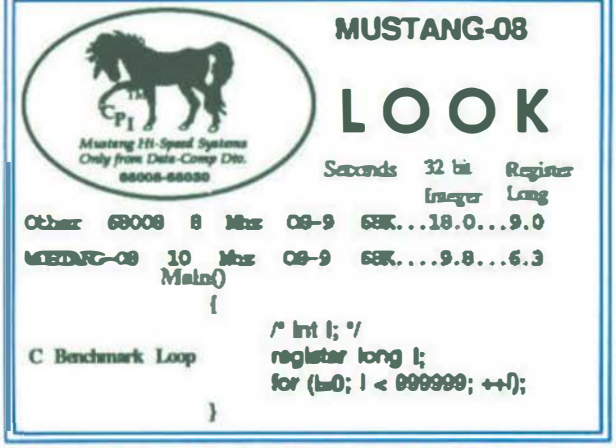
System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K No Wait States	256K Chips
PORTS	4 - RS232 2 - 8 bit Parallel	MC68681 DUART MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details



Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K Hard Disk	
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68008 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec

25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable RDM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC MUSTANG-08 is a trademark of CPI

Data-Comp Division

A Decade of Quality Service"

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC Hi-density FLEX 5" - Call for special info.